WL-TR-97-1163

SYSTEM LEVEL DESIGN METHODOLOGY FOR EMBEDDED SIGNAL PROCESSORS

EDWARD A. LEE

UNIVERSITY OF CALIFORNIA AT BERKELEY
DEPARTMENT OF ELECTRICAL AND COMPUTER SCIENCE
BERKELEY, CA 94720

AUGUST 1997

FINAL REPORT FOR 09/15/93-06/30/97

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility nor any obligation whatsoever. The fact that the Government may have formulated or in anyway supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the national technical information service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

KERRY HILL, Project Engineer
Embedded Information Systems
Engineering Branch
AFRL/IFTA

JAMES S. WILLIAMSON, Chief
Embedded Information Systems
Engineering Branch
AFRL/IFTA

STANLEY E. WAGNER, Chief
Wright Site Coordinator
Information Directorate
AFRL/IFW

IF YOUR ADDRESS HAS CHANGED, IF YOU WISH TO BE REMOVED FROM OUR MAILING LIST, OR IF THE ADDRESSEE IS NO LONGER EMPLOYED BY YOUR ORGANIZATION, PLEASE NOTIFY AFRL/IFTA, WRIGHT-PATTERSON AFB OH 45433-7334 TO HELP US MAINTAIN A CURRENT MAILING LIST.

COPIES OF THIS REPORT SHOULD NOT BE RETURNED UNLESS RETURN IS REQUIRED BY SECURITY CONSIDERATIONS, CONTRACTUAL OBLIGATIONS, OR NOTICE ON A SPECIFIC DOCUMENT.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** 8/23/97	**3. REPORT TYPE AND DATES COVERED** Final 9/15/93-6/30/97

4. TITLE AND SUBTITLE
System-Level Design Methodology for Embedded Signal Processors

5. FUNDING NUMBERS
C F33615-93-C-1317
PE 63739
PR A268
TA 02
WU 04

6. AUTHOR(S)
Edward A. Lee

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Regents of the Univesity of California
336 Sproul Hall
U.C. Berkeley
Berkeley, CA 94720

8. PERFORMING ORGANIZATION REPORT NUMBER
442427-25327

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Avionics Directorate
Wright Laboratory
Air Force Materiel Command
Wright-Patterson AFB OH 45433-7623
POC: Kerry L. Hill, AFRL/IFTA 937-255-7698 x3604

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
WL-TR-97-1163

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release: Distribution is Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The focus of this project was on desing methodology for complex real-time, reactive systems where a variety of design methodologies and implementation technologies must be combined. Design methodologies are encapsulated in one or more models of computation, while implementatiuon technologies are implemented as synthesis tools. Applications that use more than one model of computation and/or more than one synthesis tool are said to be heterogeneous. Hardware/ software codesign is one example of such heterogeneous design.

14. SUBJECT TERMS
VHDL, design methodology, heterogeneous design

15. NUMBER OF PAGES
72

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

TABLE OF CONTENTS

1. Project Overview

The focus of this project was on design methodology for complex real-time, reactive systems where a variety of design methodologies and implementation technologies must be combined. Design methodologies are encapsulated in one or more *models of computation*, while implementation technologies are implemented as synthesis tools. Applications that use more than one model of computation and/or more than one synthesis tool are said to be *heterogeneous*. Hardware/software codesign is one example of heterogeneous design.

The project developed formal models for such heterogeneous systems, a software environment for the design of such systems, and synthesis technologies for implementation of such systems. In the latter category, it concentrated on problems not already addressed well elsewhere, such as the synthesis of embedded software (*code generation*, sometimes called *auto-coding*) and the partitioning and scheduling of heterogeneous parallel systems.

2. Heterogeneous Design Principles

2.1 REACTIVE SYSTEMS

Many traditional computational systems are *transformational*, in that they transform a body of input data into a body of output data. Operating systems and network-aware applications, such as those with a client-server architecture, are *interactive*, in that they interact with the environment, but they interact at their own speed. This project was concerned with systems that are *reactive*, in that they react continuously at the speed of the environment. It focused primarily on a subset of such systems, those with a large component of signal processing. Such systems are computationally intensive, hard-real-time, and typically embedded and concurrent.

2.2 SYSTEM-LEVEL DESIGN

By "system-level design" we mean design at the problem level that is relatively unencumbered by implementation issues. For signal processing applications, a block-diagram style of specification and design is popular, primarily because it matches the applications well. A typical model of a system, implemented in Ptolemy, is shown in figure 1.

Such specifications are *modular*, in that large designs are composed of smaller designs, and these smaller designs encapsulate specialized expertise. They are *hierarchical*, in that composite designs themselves become modules, and modules may be very complicated. They are *concurrent*, in that modules logically operate simultaneously. Implementations may be sequential, parallel, or distributed. They are *abstract*, in that the interaction of modules occurs within a model of computation. They are *domain specific*, tuned in this case to the needs of signal processing applications. Often they will need to combine multiple domain-specific subsystems.

To be successful, system-level design must be coupled with high quality synthesis tools that translate system-level specifications into implementations. For signal processing, dataflow models of computation provide a convenient and popular means for specification. Thus, much of our work focused on the syntax and semantics of such specifications and the synthesis of implementations from them.

2.3 HETEROGENEOUS IMPLEMENTATIONS

Embedded reactive systems today are typically implemented using a combination of implementation technologies, as suggested in figure 2. Custom digital hardware, for example, may be combined with analog, microwave, or microelectromechanical systems (MEMS) designs. Hard real-time software, written in assembly code for a specialized processor like a programmable DSP, may be combined with higher-level software, typically written in C, that implements the control logic of the

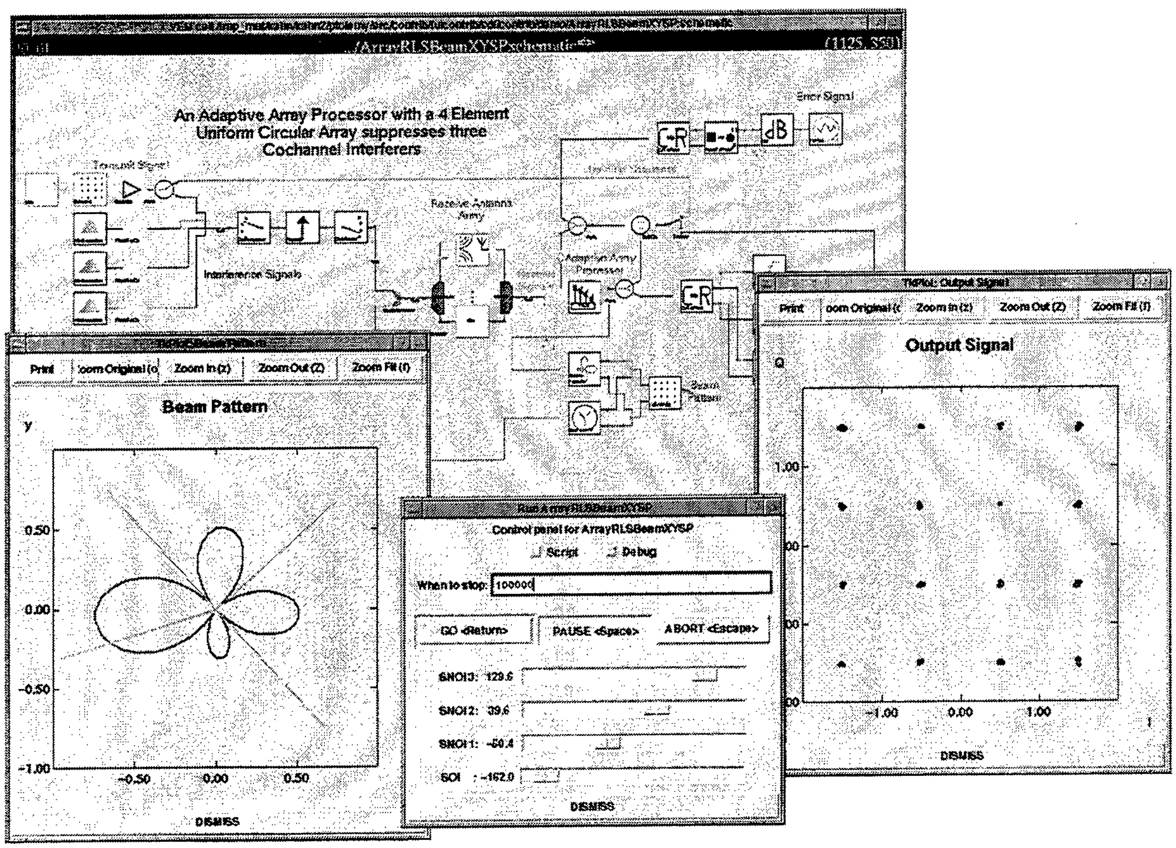

FIGURE 1. A Ptolemy application, developed by an outside Ptolemy user (Uwe Trautwein of the Technical University of Ilmenau, Germany), depicts a beam former that adaptively nulls interferers. It is an interactive, animated simulation, where on-screen controls modify the direction of arrival of the signal, and uses the higher-order functions and Tcl/Tk scripting capabilities in Ptolemy.

application. And of course, hardware and software are combined within the same design.

Two competing approaches to the design of such systems are the *grand unified approach* and the *heterogeneous approach.* The grand unified approach seeks to find a common representation language for all components, and to develop techniques to synthesize diverse implementations from this representation. The heterogeneous approach uses domain-specific models of computation hierarchically mixed and matched to define a system and seeks to find retargettable synthesis techniques from specifications to diverse implementation technologies. This project pursued the latter approach, and we believe that the results demonstrate the validity of the approach.

The heterogeneous approach has a number of advantages. First and foremost, it is clearly possible,

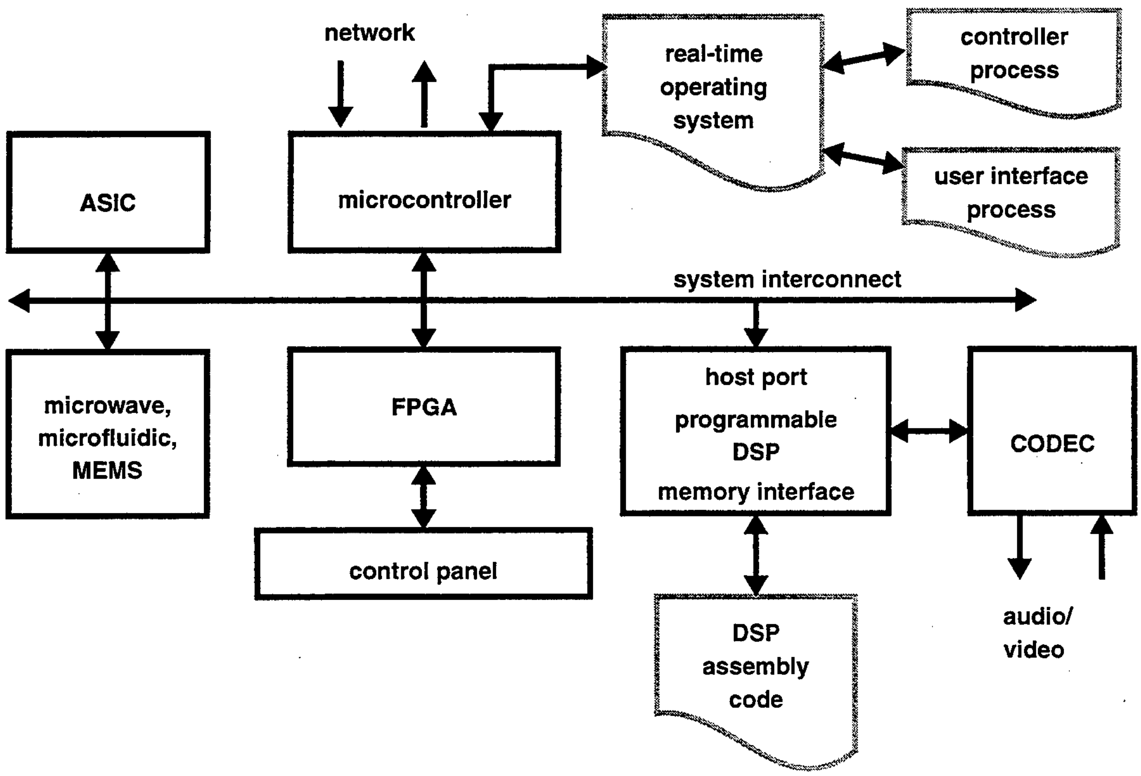

FIGURE 2. Typical hardware architecture for an embedded reactive system with a significant signal processing component. The architecture is highly heterogeneous, and its hardware-software combination is only one manifestation of this.

while there is no clearly usable grand unified approach. In addition, it emphasizes domain specific techniques, which match the applications better. Furthermore, because they are more specialized, domain-specific techniques are more amenable to high-level synthesis.

Any particular (known) candidate for a grand unified approach has a number of serious disadvantages. First, it must, of necessity, impose a model of computation. For example, choosing to use an imperative language will impose a sequential model of computation. But any particular model of computation can greatly affect the chosen system architecture. Using an imperative language, for instance, will strongly bias implementations towards software over hardware. On the other hand, using a discrete-event model of computation, as with structural VHDL, will strongly bias the implementation towards hardware over software. If a grand unified approach fails to impose a model of computation, then it will have all of the disadvantages of the heterogeneous approach and none of the advantages.

In the heterogeneous approach, multiple models of computation may be used at the problem level (figure 1) and the implementation level (figure 2). The core of the project, therefore, was on the relationship between heterogeneous models at these two levels, as suggested in figure 3. This relationship consists of a modeling relationship (where a problem-level description is a model of an implementation), synthesis (where a problem-level description is translated into an implementation-level description), and mapping (where modules at one level are related to modules at the other).

2.4 MODELS OF COMPUTATION

There are a rich variety of models of computation that deal with concurrency in different ways. In this section, we outline some of the most promising models that we uncovered during the course of this project. All of these will lend an interpretation, or *semantics*, to the same bubble-and-arc, or block-and-arrow diagram shown in figure 4.

2.4.1 Differential Equations

One possible semantics for the syntax in figure 4 is that of differential equations. The arcs represent continuous functions of a continuum that is interpreted as time. The bubbles represent relations between these functions. The job of a simulator is to find a fixed-point, i.e., a set of functions that sat-

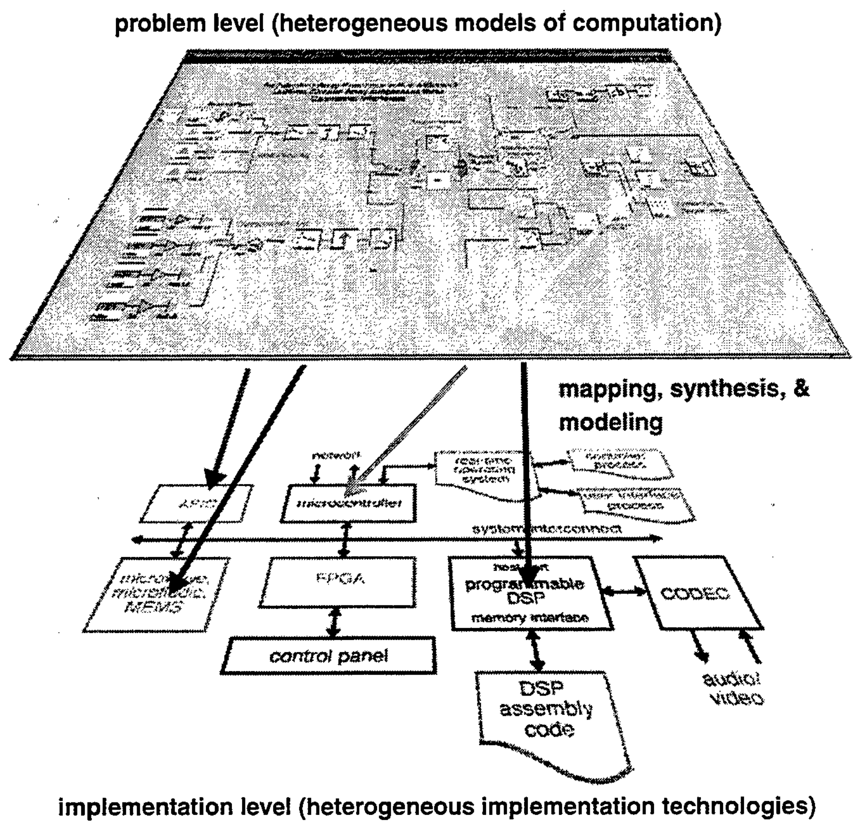

FIGURE 3. The focus of this project was on heterogeneous problem-level modeling, heterogenous implementation-level modeling, and the relationship between these levels.

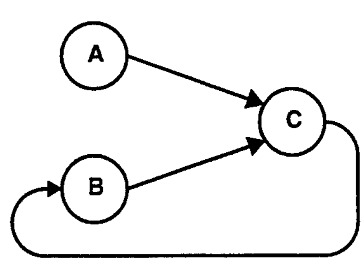

FIGURE 4. A single *syntax* (bubble-and-arc or block-and-arrow diagram) can have a number of possible *semantics* (interpretations).

isfy all the relations.

Differential equations are excellent for modeling analog circuits and many physical systems. This is the model of computation used in Spice circuit simulators. However, they have disadvantages. Since they directly describe a physical system, they are tightly bound to an implementation, leaving few implementation options. In addition, they can be expensive to simulate (and hence, expensive to implement in software). Thus, differential equations are best used for implementation-level modeling.

Although at Berkeley we have not created the ability to use differential equations in Ptolemy, Hewlett-Packard has recently announced an integration of Ptolemy with their well regarded analog and microwave circuit simulators. For more details, see Section 3 "Technology Transfer" on page 18.

2.4.2 Difference Equations

Differential equations can be discretized to get difference equations, a commonly used model of computation in digital signal processing. This model of computation can be further generalized to support multirate difference equations. In either case, a global *clock* defines the discrete points at which signals have values (at the *ticks*).

Difference equations are considerably easier to implement in software, and hence leave more freedom of implementation. Thus, they can be used at the problem level. Their key weaknesses are the global synchronization implied by the clock, and the awkwardness of specifying irregularly timed events and control logic.

The *synchronous dataflow* domain in Ptolemy is used to model difference equations, although it is slightly more general, and avoids the global synchronization implied by a pure interpretation of difference equations.

2.4.3 Process Networks and Dataflow

In a Process Network (PN) model of computation, the arcs represent sequences of data values (tokens), and the bubbles represent functions that map input sequences into output sequences. Certain technical restrictions on these functions are necessary to ensure *determinacy*, meaning that the sequences are fully specified. Dataflow models, popular in signal processing, are a special case of process networks [14].

PN models are excellent for signal processing. They are loosely coupled, and hence relatively easily to parallelize or distribute. They can be implemented efficiently in both software and hardware (something demonstrated by this project), and hence leave many implementation options open. Thus, they are best used for problem-level specification.

A key weakness of PN models is that they are awkward for specifying control logic.

PN models are implemented in Ptolemy using a hierarchy of four nested domains. These are, from smallest (least general) to largest (most general): *synchronous dataflow* (SDF), *boolean dataflow* (BDF), *dynamic dataflow* (DDF), and *process networks* (PN). Many improvements in this technology were completed under this project, and many of the results have been successfully transferred to industry.

2.4.4 Synchronous/reactive Models

In the Synchronous/Reactive (SR) model of computation, the arcs represent data values that are aligned with global clock ticks. Thus, they are discrete signals, as with difference equations, but unlike difference equations, a signal need not have a value at every clock tick. The bubbles represent relations between input and output values at each tick, and are usually partial functions with certain technical restrictions to ensure determinacy.

SR models are excellent for applications with concurrent and complex control logic. They can be realized in the popular Esterel language and certain variants of the Statecharts language. Because of the tight synchronization, however, some applications are overspecified in the SR model, limiting the implementation alternatives. Moreover, in most realizations, modularity is compromised by the need to seek a global fixed point at each clock tick.

A key result of this project was to develop a modular SR model of computation and an implementation in Ptolemy [74]. This is the first realization of the SR model of computation that mixes cleanly with other models of computation, thus allowing the use of SR for control logic in combination with, for example, dataflow for signal processing.

2.4.5 Discrete-Event Models

In discrete-event (DE) models of computation, the arcs represent sets of *events* placed in time. An event consists of a *value* and *time stamp*. This model of computation is popular for specifying hardware and simulating telecommunications systems, and has been realized in a large number of simulation environments, simulation languages, and hardware description languages, including VHDL and Verilog. Unlike the SR model, there is no global clock tick, but like the SR, differential equations, and difference equations, there is globally consistent notion of time.

DE models are excellent descriptions of concurrent hardware, although increasingly the globally consistent notion of time is problematic. In particular, it over-specifies (or over-models) systems where maintaining such a globally consistent notion is difficult, including large VLSI chips with high clock rates. A key weakness is that it is relatively expensive to implement in software, as evidenced by the relatively slow simulators.

2.4.6 Rendezvous Models

In a rendezvous model, the arcs represent sequences of atomic exchanges of data between sequential processes, where the bubbles represent the processes. "Atomic" means that the two processes are simultaneously involved in the exchange. Examples of rendezvous models include Hoare's *communicating sequential processes* (CSP) and Milner's *calculus of communicating systems* (CCS). This model of computation has been realized in a number of concurrent programming languages, including Lotos and Occam.

Rendezvous models are particularly well-matched to applications where resource sharing is a key element, for example, client-server database models. A key weakness of rendezvous-based models is that maintaining determinacy can be difficult. Proponents of the approach, of course, cite the ability to model nondeterminacy as a key strength. We have not (yet) implemented a domain in Ptolemy supporting the rendezvous style of concurrency because it did not seem to match the needs of RASSP well.

2.4.7 Finite-State Machines

In FSMs, bubbles represent system *state* and arcs represent state *transitions*. This model of computation is radically different from all the previous ones in that it is not concurrent. Execution is a strictly ordered sequence of state transitions.

FSM models are excellent for control logic in embedded systems, particular safety-critical systems. FSM models are amenable to in-depth formal analysis, and thus can be used to avoid surprising behavior. Moreover, FSMs are easily mapped to either hardware or software implementations, and thus are suitable for use at the problem level.

FSM models have a number of key weaknesses. First, at a very fundamental level, they are not as expressive as the other models of computation described here. They are not sufficiently rich to describe all partially recursive functions. However, this weakness is acceptable in light of the formal analysis that becomes possible. Many questions about designs are decidable for FSMs and undecidable for other models of computation. A second key weakness is that the number of states can get very large even in the face of only modest complexity. This makes the models unwieldy.

The latter problem, however, can be solved by using FSMs in combination with concurrent models of computation. This was first noted by David Harel, who introduced that Statecharts formalism, which combines a loose version of SR with FSMs. FSMs have also been combined with differential equations, yielding the so-called *hybrid systems* model of computation.

A major result of the Ptolemy project has been to show that FSMs can be hierarchically combined with all of the concurrent models of computation described above. We call the resulting formalism "*charts" (pronounced "starcharts") where the star represents a wild card. This result came fairly late in the program, and thus has not been completely implemented in Ptolemy. However, combinations of FSM with synchronous dataflow and discrete-event were implemented and released. Part of this work was done with additional funding from Lockheed-Martin.

2.5 CHOOSING MODELS OF COMPUTATION

The rich variety of available concurrent models of computation outlined in the previous section can be daunting to a designer faced with having to select them. Most designers today do not face this choice because they get exposed to only one or two. This is changing, however, as the level of abstraction and domain-specificity of design software both rise.

An essential difference between concurrent models of computation is their modeling of time. Some are very explicit by taking time to be a real number that advances, and placing events on a time line or evolving continuous signals along the time line. Others are more abstract and take time to be discrete. Others are still more abstract and take time to be merely a constraint imposed by causality. This latter interpretation results in time that is partially ordered, and explains much of the expressiveness in process networks and rendezvous models of computation. Partially ordered time provides a mathematical framework for formally analyzing and comparing models of computation. This observation has led to some key theoretical results under this project [66]. These results have profoundly affected our view of Ptolemy domains and their interrelationships.

A grand unified approach would seek a concurrent model of computation that serves all purposes.

This could be accomplished by creating a *melange*, a mixture of all of the above, but such a mixture would be extremely difficult to use, and synthesis and simulation tools would be difficult to design. Another alternative would be to choose one concurrent model of computation, say the rendezvous model, and show that all the others are subsumed as special cases. This is easy to do. Most of these models of computation are sufficiently expressive to be able to subsume most of the others. However, this fails to acknowledge the strengths and weaknesses of each model of computation.

We believe that a key result of this project is to show that the heterogeneous approach is viable and much more promising than the grand unified approach, at least in the near term. This result follows from considering the problem of validating designs. Validating designs consists of verifying that certain desirable properties are held and that certain undesirable properties are not held.

Perhaps the most important consideration in choosing a model of computation is the impact that the choice has on the quality of the end design. Two key aspects of this quality are its *correctness* and its *cost*. Let us focus on correctness. The choice of model of computation can strongly affect the ability to validate the correctness of a design.

A number of methods can be used to validate designs. The simplest to use is validation *by construction*, where the property to be verified is true of all designs specified within the model of computation, so a particular design does not need to be explicitly validated. An example of a property that is often verified this way is determinacy, which loosely means that the specification completely describes the behavior of a system. A model of computation that yields to this style of validation is intrinsically limited in expressiveness, since it cannot describe designs that violate certain properties.

In practice, few properties can be practically validated by construction. A second choice would be *formal verification*, where a property is verified by either formal manipulation of the syntax of the specification or by algorithmic search over possible behaviors. Formal verification, however, has proven practical only with models of computation with rather limited expressiveness, such as finite state machines.

A third choice is to validate designs by *simulation*, in which a property is shown to hold over a set of example inputs. In practice, it is difficult (or often impossible) for the set of example inputs to be comprehensive, representing all possible inputs. Thus, this validation method is less reliable than the prior methods.

An alternative to simulation is validation by *prototyping*, where a representative implementation of the system to be validated is built and deployed in a representative environment. This method often permits more exhaustive testing, although in practice, the representative environment may fail to represent a realizable environment that will cause the system to fail.

The last resort is validation by *intuition*. In practice, some combination of intuition, simulation, and prototyping is the most common form of validation today. The role of intuition is critical, and indeed exploits the considerable strengths of the human abilities of the designers. However, as system complexity increases, intuition breaks down.

The validation methods described are not all equally desirable. In the order given here (by construction, formal verification, simulation, prototyping, and intuition), we would argue that if a property can be verified by a technique earlier in the list, then a designer should always choose to use that method over a method later in the list. Thus, the choice of model of computation should be influenced by the desire to move validation up the list.

Validation methods early in the list, however, are more effective if the model of computation is more limited in expressiveness. Thus, the desire to work with an expressive and general model of computation is at odds with the effectiveness of validation.

A similar argument can be made for synthesis. Effective synthesis requires more restricted models of computation. A familiar form of evidence for this argument is that VHDL has been written using a particular style in order for hardware synthesis tools to generate cost-effective designs. The expressiveness of VHDL is excessive from this perspective. While it is possible that this reflects limitations in the state of knowledge, we believe that instead it reflects fundamental limitations. Effective synthesis from high-level descriptions requires that the high-level descriptions exist within a model of computation that has limited expressiveness.

This intrinsic tension, between expressiveness and validation/synthesis, can only be resolved through heterogeneity. Systems must be broken into modules, and these modules must be designed within specialized models of computation that match their functionality. This is the key principle underlying the Ptolemy project.

3. Technology Transfer

One of the notable successes of this project was its demonstrable transfer of technology to industry leaders in the computer-aided design and defense industries. This was accomplished via a careful cultivation of industry contacts and a strategy of wide open, very liberal distribution of software and publications. All software was made available on the Web with the most liberal copyright notice permitted by the University of California. This notice retains ownership of the copyright, but expressly grants permission to use the software for any purpose, including development of commercial products. It is distinctly more liberal than the GNU public license, and thus better represents "free software."

Two leaders in the CAD industry, Cadence (The Alta Group) and Hewlett-Packard (EEsof Division) made use of this policy to incorporate technology from this project into their CAD frameworks, as discussed below. Several smaller companies made use of Ptolemy software, extending it for their own purposes, and numerous companies used Ptolemy as an experimental design environment.

Efforts to further promote technology transfer included the development of a new graduate class at Berkeley, the organization of two miniconferences, and the hosting of a visiting scholar from one of the leading government labs working in system level design, the group at the Naval Research Labs that has developed the *processing graph method* (PGM), which is closely related to the dataflow technology advanced in the Ptolemy project.

3.1 CADENCE USES PTOLEMY IN SPW3.5

On October 23, 1995, The Alta Group of Cadence Design Systems announced SPW 3.5, which contains three key technologies from Ptolemy: mixing of discrete-event and dataflow models of computation, and synchronous and dynamic dataflow scheduling technology. The subtitle of Alta's press release is:

"New SPW* Simulation Technology for Convergence Applications Leverages Berkeley's Ptolemy Project Research"

In the body of the press release:

"The new simulation architecture is based on research from the renowned Ptolemy research project at the University of California at Berkeley. ... [It] utilized the Ptolemy team's results to uniquely implement Ptolemy's advanced simulation algorithms in Alta Group's leading SPW solution."

We believe that this interaction with Cadence and others has ensured that the best results of the project make their way into self-sustaining commercial products. The full press release is available on the Ptolemy Web site.

3.2 HEWLETT-PACKARD INTEGRATES PTOLEMY WITH ANALOG SIMULATION

On June 2, 1997, Hewlett-Packard's EEsof Division announced plans to deliver a comprehensive digital signal processing (DSP) design system as part of its effort to broaden its solutions for the electronic design automation (EDA) industry. In their June 2, 1997, press release, HP EEsof states:

"Built into the HP DSP Designer software is a new simulation technology developed by merging HP research and technology with the University of California at Berkeley Ptolemy project. This new simulation engine facilitates cosimulation of time, frequency and data flow technologies and significantly expands the DSP development capability for mixed RF/analog/DSP communications projects."

The software is comprised of two new DSP tools - DSP Designer and DSP Synthesis. It is part of HP's newly introduced HP Advanced Design System, which includes the latest versions of its highly regarded RF and analog circuit simulation technology. The complete press release and a related article from *EE Times* are available on the Ptolemy Web site.

3.3 LOCKHEED-MARTIN DEVELOPS ARCHITECTURAL TRADE-OFF ANALYSIS TOOL

Sanders, a Lockheed-Martin company, has been using Ptolemy to develop tools for architectural evaluation and trade-off analysis. Their work leverages the SDF and DE domains in Ptolemy, enhanced with their own user interface and visualization tools.

3.4 BNED EXTENDS PTOLEMY FOR MODELING TELECOMMUNICATION NETWORKS

We have recently learned about the establishment of a new company in Germany that develops products based upon Ptolemy. BNeD, Broadband Network Design, develops and sells products based upon Ptolemy for analysis, planning, optimization and testing of next generation telecommunication core (optical) and local exchange networks. Their Web page is:

http://www.bned.com

3.5 DQDT USES PTOLEMY VHDL GENERATION FOR ASIC DESIGN

DQDT, Dimensions in Quick Design Turnaround, derived a new VHDL domain in Ptolemy to serve as a front end specification and VHDL code generation environment for behavior modeling and synthesis of ASICs.

3.6 BDTI USES THE PTOLEMY KERNEL TO INTEGRATE OTHER TOOLS

Berkeley Design Technology Inc. (BDTI) wrote a layer on top of the Ptolemy kernel called Ptolemy HSIM (Heterogeneous Simulation) to serve as a simulation backplane that allowed Cadence's Signal Processing Workstation (SPW), Cadence's Bones and Precedence's SimMatrix tools to cooperate during a simulation. (Precedence has since been acquired by Mentor Graphics.) SimMatrix is a synchronization mechanism for connecting 30 different VHDL and Verilog simulators together.

3.7 TECHNOLOGIES LYRE DEVELOPS DSP RAPID PROTOTYPING UNDER PTOLEMY

Technologies Lyre, in Québec City, Canada has recently developed a rapid-prototyping DSP development platform that works under Ptolemy and MatLab. Contact Jean-Francois Ouellet, Technologies Lyre, aad902@agora.ulaval.ca.

3.8 PTOLEMY MINICONFERENCES

We held two miniconferences at Berkeley that reviewed major accomplishments of the Ptolemy project. The objectives of the conference were primarily to report to and solicit advice from the industrial sponsors and friends of the project. Both miniconferences were held in conjunction with Berkeley's annual Industrial Liaison Program (ILP) conferences, which included two afternoons of laboratory demonstrations of Ptolemy software prior to the miniconferences.

3.8.1 First Ptolemy Miniconference — March 10, 1995

The First Ptolemy Miniconference drew 50 sponsors and friends of the Ptolemy project from the following organizations:

- ARPA/ESTO
- Berkeley Design Technology
- Cadence (the Alta Group)
- Dataflow Systems
- Ericsson
- Hewlett-Packard
- Hitachi
- Hughes Aircraft Company
- Lockheed Sanders
- Mercury Computer Systems
- Mitsubishi
- Motorola (three separate groups)
- NASA
- Rockwell
- Semiconductor Research Corporation (SRC)
- Sony
- Synopsys
- Thomson CSF
- United States Air Force
- UniView Systems
- Westinghouse
- White Eagle Systems Technology
- Wind River Systems

Presentations at the conference included:

- An Overview of the Ptolemy Kernel Architecture.
- Design Methodology Management for System-level Design.

- Symbolic Computation in System Simulation and Design.
- VHDL Code Generation for Simulation and Synthesis.
- Optimization Issues in Embedded Software Synthesis.
- Combined Code and Data Memory Minimization.
- Parallel Implementation.
- Real-Time Prototyping.
- Mixing Dataflow with Control.
- An Introduction to a Mathematical Model of Dataflow.
- The Process Network Domain.

3.8.2 Second Ptolemy Miniconference — March 14, 1997

This miniconference reviewed both this DARPA effort, which was at the stage of wrapping up, and future plans and preliminary results under a new DARPA effort entitled "Design of Distributed Adaptive Signal Processing Systems." The conference included several outside speakers reporting on uses of Ptolemy software and techniques plus ongoing interactions. We had 58 attendees from the following organizations:

- Adaptec
- Advanced Fibre Communications
- Advantest
- Alta Group of Cadence Design Systems
- Angeles Design Systems
- Berkeley Design Technology
- Data Flow Systems
- Ericsson Radio Systems AB
- Hewlett Packard
- Hughes Aircraft
- Hughes Space and Communications
- LG Electronics
- Lockheed-Martin
- Motorola
- National Semiconductor
- NEC
- Nortel
- Rockwell International
- Sanders, a Lockheed Martin Company
- Seiko Epson Corp.
- Semiconductor Research Corporation

- Seoul National University
- Sony
- Structured Software Systems
- Sun Microsystems
- Synopsys
- Tektronix
- Thomson-CSF
- University of Pittsburg
- University of Texas, Austin
- University of Washington
- White Eagle Systems

The highlights of the conference included:

- The first public demonstration of hierarchical finite-state machines combined with dataflow and discrete-event concurrency models.
- The first public demonstration of a synchronous/reactive modeling environment that supports hierarchical heterogeneity.
- The first public demonstration of Tycho, our user-interface development environment, interacting with Java and with Ptolemy.
- The first public demonstration of Web-based simulators for programmable DSPs, from UT Austin.
- The first public description of an investment analysis tool from Structured Software Systems, based on Ptolemy.

The miniconference also included descriptions of the use of Ptolemy in modeling free-space optoelectronic systems (from the University of Pittsburg), a description of Myrnet network simulations in Ptolemy (from Sanders), the use of Ptolemy for VHDL-based circuit design, research on multidimensional signal processing models, and theory that we have developed to help us understand interacting models of computation. In addition, we outlined plans for future work including a strategy for supporting fixed-point design and our plans for Java-based design. The proceedings of the conference are at:

http://ptolemy.eecs.berkeley.edu/papers/viewgraphs/miniconf97/

3.9 PTOLEMY TUTORIAL

In conjunction with Dave Wilson of Berkeley Design Technology, Mike Williamson, Brian Evans, and Edward Lee led a full-day tutorial on Ptolemy at the RASSP conference in Arlington Virginia in

1995. Approximately 25 people attended.

3.10 OTHER DISTRIBUTION MECHANISMS

SAL (Scientific Applications on Linux) includes Ptolemy version 0.6 on a CD ROM and Web site. The URL is:

http://SAL.KachinaTech.COM/

Ptolemy is located at the following URL:

http://SAL.KachinaTech.COM/E/1/PTOLEMY.html

Also, Ptolemy is now listed under Yahoo. The link is:

http://www.yahoo.com/Science/Computer_Science/
Electronic_Computer_Aided_Design__ECAD_/Tools/

3.11 NEW BOOK: SOFTWARE SYNTHESIS FROM DATAFLOW GRAPHS

A new book [1] studies the problem of synthesizing software for embedded signal processing systems starting from applications expressed as synchronous dataflow (SDF) graphs. After a comprehensive review of the theory behind SDF, techniques are given to optimize primarily the program memory size and secondarily the data memory size. To accomplish this, SDF graphs describing multirate signal processing applications are scheduled into nested loops. A formal theory for constructing and manipulating these loops is developed, and a class of looping structures, called single appearance schedules, is shown to be the most efficient with respect to code size. The existence of such structures is studied, and algorithms for optimally constructing them are given. Extensive experimental data is presented, demonstrating the efficacy of the techniques.

3.12 POLIS — A CODESIGN SYSTEM BASED ON PTOLEMY

The group of Prof. Alberto Sangiovanni-Vincentelli at Berkeley has released a Ptolemy-based codesign environment for control-dominated embedded systems, called POLIS. POLIS offers an integrated interactive environment for specification, cosimulation, formal verification, and synthesis of embedded systems implemented as a mix of hardware and software components. It uses and significantly extends the discrete-event (DE) domain in Ptolemy. See:

http://www-cad.eecs.berkeley.edu/Respep/Research/hsc/abstract.html

Cadence is known to be heavily influenced by Polis and may be commercializing it.

3.13 A NEW GRADUATE CLASS ON MODELING OF SYSTEMS

We organized a new graduate class, EE290N, "Specification and Modeling of Reactive Real-Time Systems." This class incorporated recent results obtained under this project, and is likely to become (after further evolution) a regular graduate class. The description of the class follows:

> "This research seminar studies models of computation and programming language semantics used for the specification and modeling of real-time and reactive electronic systems. It begins with a review of the theory of partially ordered sets, particularly as applied to prefix orders and Scott orders. It develops a framework for models of computation for concurrent systems that uses partially ordered tags associated with events. Discrete-event models, synchronous/reactive languages, and dataflow models are studied in this context. Basic issues of Turing completeness and lambda computability, boundedness, determinacy, reachability, and liveness are studied, with emphasis on decidability and efficiency of verification and synthesis algorithms. Classes of functions over partial orders, including continuous, monotonic, stable, and sequential functions are considered. A hierarchy of increasingly specialized asynchronous models, including process networks, Kahn process networks, dataflow process networks, the Boolean dataflow model, and the synchronous dataflow are covered. Timed models, including discrete-event systems (as embodied for example in the VHDL and Verilog languages) and the synchronous/reactive languages Signal, Lustre, Esterel, and Statecharts are studied. Throughout, applications to signal processing, real-time, and reactive systems are emphasized, as are synthesis and compilation techniques amenable to such modern approaches as embedded system design, hardware/software codesign and formal verification."

An early version of this class was reported in [43]. More information about the most recent version of the class can be found at its Web site:

http://www.eecs.berkeley.edu/~eal/ee290n/

3.14 EMBEDDED SOFTWARE SYSTEMS CLASS AT UT AUSTIN

Professor Brian Evans, formerly a postdoc under this project at Berkeley, introduced a new course at UT Austin entitled "Embedded Software Systems" that is based on the system-level design issues tackled under this project. The course uses Ptolemy for demonstrations, homework exercises, and student projects, leverages material from the graduate class described in the previous subsection, and uses the book that summarizes many of the results of this project [1]. It its most recent offering, the course featured two guest speakers from the Ptolemy project (Praveen Murthy and Stephen Edwards). All notes, handounts, demonstrations, etc., from the class are online at

http://www.ece.utexas.edu/~bevans/courses/ee382c/

4. Summary of Accomplishments

The major accomplishments of the project are summarized in this section. Concrete deliverables included monthly and annual reports, Ptolemy software, major demonstrable technology transfer, and 86 publications, the vast majority of which have been posted on the World Wide Web. The publications consist of one book, one chapter, 14 journal articles, 41 conference papers, 6 Ph.D. theses, 6 masters reports, and 3 newsletter articles. For greater detail than in this report, refer to the publications and software at the Ptolemy Web site,

http://ptolemy.eecs.berkeley.edu/

4.1 SYSTEM-LEVEL DESIGN

System-level design in Ptolemy is concerned with issues of mapping problem-level specifications into implementations. This includes hardware/software partitioning, cosimulation, and more generally, heterogeneous simulation. It is also concerned with coupling problem-level specifications with hardware synthesis tools, including VHDL-based tools and more experimental high-level synthesis tools. It is also concerned with the manipulation of design specifications for optimization and the semantics of the problem-level specification languages.

4.1.1 Hardware/Software Partitioning

We developed and implemented a sophisticated hardware/software partitioning algorithm. This algorithm supports selection from among multiple implementations within the hardware or software categories. The area of a node implemented in hardware depends on the time allocated to run it. In our early partitioning work we assumed the hardware to be executed in the critical time (i.e., best case, corresponding to the largest area) and made a binary choice for each node, choosing either hardware or software. More recent techniques select the appropriate implementation for a node, given its area-time curve, rather than just deciding whether it is in hardware or software. Thus, instead of only solving the binary hardware/software partitioning problem, we solve the m-ary problem of partitioning into m implementation styles.

The m-ary algorithm used the binary algorithm as the core. Experiments yielded impressive results. The algorithm has complexity O(n^3), where n is the number of nodes. For an eight-node example, the optimal solution using integer linear programming required 3.5 hours. Our algorithm got close to this optimal solution and completed in 3 minutes. This work is reported in the Ph.D. thesis of Asawaree Kalavade [75].

4.1.2 Synthesis of VHDL From Dataflow Graphs

We designed a mechanism for the synthesis of VHDL from dataflow graphs. This mechanism can generate any of several different styles of VHDL code, customizing the code to optimize for synthesis by various back-end tools, or to optimize for simulation. For simulation, sequential VHDL is usually fastest. For synthesis, structural VHDL is usually most effective. We have demonstrated the translation of dataflow graphs into VHDL suitable for synthesis by the Design Compiler from Synopsys as well as rapid simulation using simulators from Synopsys and Model Technology. We completed a demonstration of a scalable beam-forming application in the retargettable VHDL domain. This uses higher-order functions (see below) to control the number of sensors. The application also has multiple sample rates. When the code generator is set to generate sequential VHDL, simulations run reasonably quickly. This work will be reported in the forthcoming Ph.D. thesis of Michael Williamson.

4.1.3 Partitioning SDF Applications Into Multiple VHDL Hardware Modules

We developed a method for the partitioning of a single application specified in synchronous dataflow (SDF) into multiple independently-synthesizable, communicating VHDL hardware modules. Either self-timed (asynchronous) or fully-static (synchronous) hardware implementations are allowed, and the clock timing and control are automatically generated. We showed that this method guarantees the preservation of correct functional behavior as specified in the original SDF graph, and that many choices of partitioning into multiple hardware modules are possible. The ability to break up a larger application into smaller synthesizable hardware modules can lead to efficiencies in hardware synthesis, which is faster when performed on smaller VHDL specifications. At the same time, the communication between the multiple modules is sufficiently specified by the method so as to ensure that the correct functional behavior is preserved when the separate modules are executed concurrently. This work will be reported in the forthcoming Ph.D. thesis of Michael Williamson.

4.1.4 VHDL-Based Hardware Design Mixed With Software and Environment Modeling

Typical systems today mix custom hardware with embedded software. Effective system-level simulation mandates inclusion of both, and in addition, a model of the environment. The principle in the Ptolemy project is to use specification, modeling, and simulation techniques that are best suited for each aspect of the design, and to mix them into a coherent whole. Thus, hardware is modeled in VHDL, embedded software in C or assembly code, and the environment at a higher, functional level.

Using our hierarchical scheduling framework (see below), we were able to get VHDL simulations to interact with Ptolemy simulations in the SDF domain (synchronous dataflow) and, more interestingly, to interact with synthesized embedded software running in C on the host processor or in assembly code on a Motorola DSP. The first demonstration system is an analysis/synthesis filter bank in which the signal stimulus and analysis of the results are done in the CGC (code generation in C) domain, the analysis half of the filter bank is done on a Motorola DSP56002, and the synthesis half of the filter bank is done in the Synopsys VHDL simulator. Both the DSP and the VHDL simulator are running code generated by Ptolemy from dataflow graphs. We believe that this is a major milestone in heterogeneous system-level design. This work is reported in [55].

4.1.5 Structural VHDL

We created two VHDL code-generation domains, called VHDLF and VHDLB. The first of these uses homogeneous synchronous dataflow semantics to describe signal processing systems at a functional level. The second uses event-driven semantics to describe arbitrary systems at the behavioral level. These domains were used successfully already in industry, by a startup company called DQDT (Dimensions in Quick Design Turnaround). VHDLF has been supplanted, however, by the more sophisticated VHDL domain described above.

4.1.6 Silage Interface to Hyper High-Level Synthesis Tool

We created a Silage domain that couples to Prof. Rabaey's high-level synthesis tool called Hyper. This domain was used for the hardware side of the hardware/software partitioning experiments conducted by Asawaree Kalavade [75]. We did not keep up this domain since Silage showed no promise of catching on as a design language.

4.1.7 Heterogeneous Simulation

With help from Prof. Soonhoi Ha of Seoul National University, Korea, we developed a clean interaction semantics for combined synchronous dataflow and discrete-event modeling. This semantics allows us to build arbitrarily deeply nested mixed systems while maintaining a consistent and intuitive notion of global time. This is challenging because the synchronous dataflow (SDF) domain has no notion of time in the conventional sense, using instead has a partially ordered notion of causality. The model we are following is that the dataflow domains appear to any timed domain to fire "instantaneously." That is, they produce outputs with the same time stamps as the inputs. If they are multirate systems, then they may optionally also produce additional events with time stamps in the future, under the control of a target parameter. The changes that were required in the software included modifications to the DE schedulers to prevent them from advancing their notion of time beyond their requested stopping time. In addition, the SDF wormhole object had to explicitly handle time stamps in order to define its multirate behavior. We have built a number of demonstration systems that illustrate this interaction. This work is reported in [9].

4.1.8 Automated Rearrangement of Signal Processing Systems

We developed and released a set of heuristic search techniques written in the Mathematica programming language. They implement breadth-first search, depth-first search, hill climbing, and simulated annealing techniques for applying a set of equivalence relationships to an algebraic expression to minimize implementation cost. One goal was to use the heuristic searches to apply the equivalence relationships in the Signal Processing Packages for Mathematica to optimize the implementations of Ptolemy systems. Both the Heuristic Search Packages and the Signal Processing Packages are available on the Ptolemy Web site.

4.1.9 Signal Reprocessing

We collaborated with the Boston University/MIT RASSP team on signal reprocessing in Ptolemy. Signal reprocessing is where, based on the output of a signal processing operation, you adjust the parameters in the operation and process the same data again to obtain a "better" result. Adaptive filtering is an example. A more complicated example concerns estimating two sinusoids of unknown spacing. One way is to use the FFT and adjust the FFT length until the sinusoids are resolved (separated).

There are a number of ways to provide a general framework for reprocessing signals using the heterogeneity supported in Ptolemy. In Ptolemy, we can define an outer reprocessing system (galaxy) that decides how to change the processing parameters in the inner dataflow subsystems (galaxies). Before firing the inner dataflow galaxies, the reprocessing galaxy would reset the parameters of the inner galaxies. The reprocessing galaxy would act as a controller of the inner galaxies. In the current release of Ptolemy, we could define the outer-level controller using the (1) dynamic dataflow domain, and (2) the synchronous dataflow domain with a higher-order function mechanism that recompiles inner galaxies before invoking them. Two new computational models are being developed and investigated to serve as outer controller systems: (1) a finite-state machine domain, at U.C. Berkeley, and (2) an integrated processing and understanding of signals domain, at Boston University and U.C. Berkeley.

At the 1994 RASSP Conference, Joseph Winograd and Hamid Nawab from Boston University demonstrated a standalone radar clutter analysis testbed using the Integrated Processing and Understanding of Signals (IPUS) architecture to process radar data using expert knowledge encapsulated by computer. This was integrated into the Ptolemy environment as an IPUS domain. The IPUS domain has a dynamic scheduler that reacts to events (knowledge) registered in global data structures (e.g.,, blackboards) by local actors (e.g., knowledge sources). The IPUS domain reasons about knowledge at different levels of abstraction arranged in a hierarchy. Various local actors (e.g. knowledge sources) have been developed that can be reused in any IPUS application.

4.2 ALGORITHM REPRESENTATION

The representation problem in Ptolemy is mainly to raise the level of abstraction to the problem level and to exploit visual syntaxes to manage complexity. Our contributions have included techniques for improving the efficacy of visual syntaxes (higher-order functions), leveraging external tools (Matlab and Mathematica, for example), and new models of computation.

4.2.1 Higher-Order Functions

We designed and implemented a higher-order functions (HOF) domain in Ptolemy that functions as a subdomain of all other domains. This has had a major impact on the usability of visual (graphical) system representations for large systems. The theory and major concepts are given in [14].

We have developed (with help from Thomson CSF) a variety of radar applications using these HOF capabilities. We believe that the resulting system representations are much more intuitive and maintainable than the traditional techniques based on multidimensional arrays (using up to seven dimensions).

4.2.2 *Leveraging External Tools*

We created a link between Matlab and Ptolemy so that stars can have their functionality expressed as Matlab functions and parameter values can be given as Matlab expressions. One major impact of this is that the full suite of graphical signal display facilities in Matlab are now available under Ptolemy. Moreover, quick algorithmic prototyping can now be done with an arbitrary mixture of Matlab (imperative, matrix-oriented) code, and block-diagram (declarative, signal-oriented) code.

A similar link was created to Mathematica, which provided the ability to include symbolic manipulations in parameter specifications. SDF demonstrations of both interface haves been released with Ptolemy since version 0.6.

4.2.3 *Communicating Processes Domain*

We developed a "communicating processes" (CP) domain in Ptolemy. This domain has been used extensively for high-level modeling of a wireless multimedia network. Unfortunately, we had to abandon this domain because it was built on top of the Sun Lightweight Process library, a fairly idiosyncratic thread library, and porting to more modern thread libraries proved difficult.

4.2.4 *Message Queue Domain*

We completed a "message queue" (MQ) domain, which is an experimental domain that models systems with highly dynamic topologies, such as telecommunications switch software.

4.3 SCHEDULING AND CODE GENERATION FOR SYNCHRONOUS DATAFLOW

Consistent with the RASSP focus on real-time signal processing in embedded systems, the Ptolemy project made several key contributions in the translation of synchronous dataflow graphs into embedded software (a technique sometimes called *auto-coding*). There are two key elements to this problem: scheduling and code generation. We made major contributions in both.

4.3.1 Scheduling of Dataflow Graphs for Efficient Synthesis

A major result of this project is a sophisticated set of scheduling algorithms that jointly minimize the size of a program and the size of data memory in embedded software generated from synchronous dataflow graphs. These algorithms and their various ramifications have been reported in a number of papers, two Ph.D. theses [73][76], and the results have been collected and published in a book [1].

4.3.2 Hierarchical Scheduling and Code Generation

We introduced a hierarchical scheduling framework that effectively mixes synthesized software and VHDL models with simulations built in other Ptolemy domains. This permits, for example, the environment to be modeled at a high level using one of the dataflow domains, while the system under design is modeled using domains that synthesize to hardware and/or software, like the VHDL and CG56 domain (the latter generates assembly code for Motorola DSPs).

We demonstrated this hierarchical scheduling on a heterogeneous platform consisting of a Sun workstation running Solaris 2.4 and a programmable DSP on an S-bus card. These demonstrations incrementally compile real-time subsystems for the DSP and embed them within a non-real-time process running on the Unix workstation. Communication between them was asynchronous, using a "peek/poke" mechanism to asynchronously read and write into the DSP memory. The demonstration systems were acoustic modems (modems that transmit from an audio loudspeaker to an audio microphone through air). Animated, interactive signal displays were produced on the workstation, enabling better evaluation and understanding of the algorithms and their performance.

The hierarchical scheduler uses common semantic properties across domains to decouple the designer-defined hierarchy (which is motivated by convenience and functional modularity) from partitioning. That is, entirely disconnected subsystems can be implemented by the same hardware module (a processor or an ASIC). This scheduling framework makes extensive use of earlier work in Ptolemy with heterogeneous multiprocessor targets.

The hierarchical scheduling mechanism permits the use of highly optimized loop scheduling techniques developed in our group. Without hierarchical scheduling, it was not possible to use these because they had not been designed for use in parallel systems. Because the applications are multirate, unless hierarchical scheduling is used, the generated code required considerably more memory than was available on the DSP card. Moreover, without hiearchical scheduling, scheduling time was substantial (because a rather large precedence graph was constructed). Thus, we demonstrated that hierarchical scheduling enables modular use of scheduling optimizations, and we have shown that in practical examples, considerable savings in embedded system memory are achieved. This work is reported in [52][53].

One of the fundamental issues encountered in this work is that dataflow models are not fundamentally compositional. Two results are reported. First, a pragmatic approach that preserves all of the advantages of current algorithms is to identify designs that happen to be compositional, and treat them as such. A sufficient condition has been reported [51][54][72]. At a more fundamental level, we have identified how dataflow models of computation can be modified to make them compositional. That work is reported in [67], but this result remains theoretical.

4.3.3 Mixing Code Generation with Simulation

We have implemented in Ptolemy an elegant and simple architecture for compiling subsystems in code generation domains and invoking them within simulation domains. There are a number of potential applications for this underlying infrastructure:

Incremental compilation. A compute-intensive subsystem in, say, the synchronous dataflow (SDF) domain can be retargeted to CGC (code generation in C) and compiled to become a single monolithic block in SDF. A similar capability is used to encapsulate a CG56 subsystem (which runs on the Motorola DSP56000) into an SDF block.

Interfacing to foreign simulators. A VHDL subsystem can be analyzed to synthesize a fast customized C interface to a commercial VHDL simulator.

Combining more than one code generation domain. For example, CGC can be mixed with CG56 to produce programs that execute concurrently on a host workstation and a DSP card.

A fundamental problem is that dataflow systems cannot always be incrementally compiled, for the same reason cited above: dataflow is not compositional. Collections of dataflow actors in a domain do not necessarily have the same semantics as an individual actor. This problem is shared by many modern languages, including all synchronous languages, such as Esterel, Statecharts, and Signal. We give fundamental results in [67], discussed further below.

4.3.4 Guided Migration: a Retargeting Tool

We developed a "retargeting tool" to be used to guide migration of Ptolemy-based designs from one implementation technology to another. We demonstrated an interface for studying differences in block libraries, and showed how it could be used to make the code generators for the Motorola DSP56000 family processors and the Texas Instruments C50 family processors more compatible. For example, Ptolemy contains demonstrations of different dual-tone multiple-frequency (DTMF) detectors that have been retargeted from the SDF simulation domain to the CGC, CG56, and C50 code generation domains. We have also developed a program that recursively changes the domain of hierarchical designs.

4.4 DATAFLOW AND KAHN PROCESS NETWORKS

Given the emphasis on dataflow modeling in signal processing circles, a natural part of the Ptolemy project was to investigate the limits on expressiveness. Dynamic dataflow and Kahn process networks are highly expressive models of computation, but pose some interesting implementation challenges. We have developed solutions.

4.4.1 Dynamic Dataflow Scheduling

A dynamic dataflow scheduler should satisfy two requirements:

- R1: If the dataflow graph does not contain a deadlock condition, the scheduler should not halt.
- R2: If the dataflow graph can be executed forever in bounded memory, then the scheduler should be able to execute it forever in bounded memory.

The latter is particularly important for embedded systems.

In general, given a dataflow graph, it is undecidable whether the graph will deadlock (the halting problem). It is also undecidable whether the graph can be executed in bounded memory (Joe Buck showed in his 1993 Ph.D. thesis how to convert this problem to the halting problem). It is easy to define a scheduling algorithm that satisfies R1 or R2, but no scheduling algorithm can always, in finite time, guarantee both R1 and R2. This problem has appeared in various forms in much of the dataflow architecture work.

In addition, the notion of an iteration in dataflow and process network domains has risen to the fore as a critical (and difficult) theoretical issue. An unambiguous definition of an iteration is necessary for control of a simulation, but even more importantly, for interaction between heterogeneous models of computation. The so-called "synchronous" methods, for example, (like statecharts and Esterel) cannot be mixed (in a determinate way) with dataflow without an unambiguous definition of an iteration. An iteration is easy to define for the synchronous dataflow (SDF) model of computation, but for dynamic dataflow and process network models, the equivalent definition fails in some cases. In particular, an iteration in SDF is a sequence of firings that returns the buffers in a dataflow graph to their original state. It is undecidable whether such an iteration exists in a dynamic dataflow or process network model.

Thus, our third condition is:

- R3: The scheduler should execute a graph in a sequence of well-defined and determinate "steps," where a step is set of actor firings.

We defined and implemented a robust and simple scheduler for the dynamic dataflow (DDF) domain in Ptolemy. It provably satisfies all three conditions.

Often, the notion of a step as defined by the scheduler is not always the notion that the user wants to see. We define an "iteration" to be one or more steps, where the number of steps is controlled by the user. To permit a user to annotate a dataflow graph with the number of firings of a block that constitute an "iteration," we implemented an extension to the GUI and the Target object to support "pragmas" attached to blocks. A given Target (such as the DDFTarget) understands only certain pragmas. In the DDF domain, the DDFTarget understands a pragma called "firingsPerIteration." Thus, when a user specifies a value of this pragma for a particular block, an "iteration" has been defined. If no such value is specified, then an "iteration" equals a "step," the scheduler default.

4.4.2 Process Networks Domain

We implemented a Process Networks (PN) domain, using first the Awesim threads package, then the gthreads package, a POSIX thread implementation from Florida State University that is distributed under the GNU General Library License. Process networks are a generalization of dynamic dataflow, and raise a number of interesting theoretical and practical issues. These issues are resolved in the Ph.D. thesis of Tom Parks [77], where it is shown that runtime scheduling algorithms exist that solve undecidable problems. In particular, there are simple algorithms that will schedule a process network in bounded memory if this is possible, without having to know a priori whether it is possible (this latter question is undecidable).

4.5 MULTIDIMENSIONAL SIGNAL PROCESSING

Dataflow models match one-dimensional signal processing extremely well. Communication between blocks (actors) is by sequences of data objects (tokens). These sequences easily represent one-dimensional discrete-time signals. However, they do not so easily represent multidimensional signals. We developed a generalization to dataflow that better matches multidimensional signal processing.

4.5.1 Multidimensional Dataflow

We have completed an experimental "multidimensional synchronous dataflow" domain (MDSDF), where arcs that connect blocks represent not simple sequences of tokens, but rather two-dimensional orderings of tokens. This domain is well matched to multidimensional signal processing and is capable of representing a broader range of algorithms with static flow of control than the synchronous dataflow model. The real potential, however, is in parallel computation, because the model of computation exposes much more parallelism at a much finer granularity than the SDF model. The domain has a rich enough set of stars to be usable for experimentation. This work is reported in [27][79].

4.5.2 Sampling Lattices

We developed a dataflow model for expressing multidimensional multirate signal processing systems sampled on arbitrary lattices. A multidimensional signal can be sampled in many different ways. A straightforward extension of one-dimensional sampling results in the so-called rectangular sampling structure, where the samples lie on a rectangular grid. However, a more general sampling structure is a geometrical lattice; sampling lattices that are not rectangular can have many advantages in certain applications. For example, a signal sampled on a nonrectangular lattice can have a lower sampling density than one sampled on an equivalent rectangular lattice. For real-time processing of multidimensional signals, a lower sampling density means fewer samples to process in a given time interval. The standard MDSDF model suffers from the inability to model multidimensional systems sampled on arbitrary sampling lattices; hence, we give an extension of MDSDF that is capable of modeling such systems. The model we give preserves the property of static, compile-time schedulability. However, constructing such schedules requires the solution to some challenging problems. In particular, we show that an augmented set of balance equations has to be solved simultaneously in the extended model. The additional equations are quite different from the usual balance equations in SDF and MDSDF; they involve computing so-called "integer volumes" of parallelepipeds. This computation turns out to be an interesting number-theoretic problem, and we present several approaches for solving it. Finally, we present a practical example of a video sampling structure conversion system to show the usefulness of the generalized MDSDF model. This work is reported in the Ph.D. thesis of Praveen Murthy [76].

4.5.3 Filter Design Issues

The design of multidimensional multirate signal processing systems. e.g., systems that change video formats in nonseparable ways, often require application-specific design tools. For example, in computing system parameters in multidimensional multirate systems can be simplified with a combination of computational geometry, integer matrix algebra, and state-space formulations. In multiple dimensions, rate-changing operations are defined by a change in sampling grids. Sampling grids can be represented as a set of basis vectors, which can be considered as the column vectors that make up a sampling matrix. Mapping one sampling matrix onto another is a linear mapping represented by a rational matrix, called a resampling matrix. We have shown how to design two-dimensional rate changing systems (upsampler, filter, and downsampler in cascade) based on a geometric sketch of the passband to retain. From the sketched region, we use computational geometric techniques to find the minimal enclosing parallelogram using a linear time and linear space algorithm we have developed. We then use the minimal enclosing parallelogram to compute the resampling matrix to perform the sampling conversion using Chen and Vaidyanathan's approach. Then, we factor the resampling matrix into the up-sampling and down-sampling matrices for the rate changer. The procedure will find the best compression rate based on a parallelogram-shaped passband. The only other admissible geometry is a hexagonal-shaped passband, which will always do at least as well as the parallelogram-shaped passband. Generalizing this approach to multiple channels will enable the graphical design of two-dimensional filter banks and wavelets. This work is reported in [35][36].

4.6 MULTIPROCESSOR TARGETS

Embedded signal processing systems often require more than one processor to meet real-time constraints. We have made some contributions in the area of automatic generation and optimization of multiprocessor implementations.

4.6.1 Resynchronization

We developed a set of algorithms for minimizing the number of synchronized communications between multiple processors in a multiprocessor system [25][61][62]. Synchronized communications are considerably more expensive than unsynchronized communication, requiring testing and setting semaphores. The algorithms are based on the observations that some synchronizations are redundant, since it can be algorithmically demonstrated that the semaphores will always be in the desired state, regardless of timing. These synchronizations can be removed. A second (complementary) method selectively adds synchronization operations that will then cause other synchronization operations to become redundant. We have proven that the problem is NP-hard, but have established a correspondence with the well-studied set-covering problem, which provides a wealth of heuristic solutions. A third method converts a feed-forward dataflow graph into a strongly connected graph in such a way as to reduce synchronization overhead without slowing down the execution. All three methods can be applied as post processing optimizations to the output of any static parallel scheduling algorithm. A more recent extension of these algorithms considers latency constraints as well, giving provably optimal algorithms. Results are described in [23][24][25][61][62][63].

4.6.2 Targeting a Network of Workstations (NOW) Cluster

We implemented a target in the CGC (code generation in C) domain that produces code for a NOW (Network of Workstations) cluster. The generated code is built on top of the active message abstraction, and hence is portable and potentially quite efficient. We have shown that the same set of parallel executables can be run on an ordinary cluster of networked workstations as well as on the specially configured NOW. Surprisingly, initial tests resulted in faster runs on the ordinary cluster, but further tuning has now achieved better performance on NOW. Currently, in the Berkeley NOW cluster, active messages are implemented on top of TCP/IP, so there is considerable communication overhead. However, as that facility matures, and this overhead is removed, we will be able to track it and improve performance.

4.6.3 Mercury Raceway Architecture

We outlined the design of a tool for mapping a control and dataflow representation of a hard real-time signal processing application onto a Mercury RACEway multicomputing system, and are continuing development with other funding. Low-level programming details would be hidden from the programmer thereby shifting the design focus to performance issues. Graphical visualization and manipulation capabilities will enable study of architectural trade-offs and optimizations. Tasks can be scheduled and partitioned among the processors either manually or automatically. The tool will handle most of the details involved in generating multiprocessor code, downloading the code to the target system, and initializing the system for execution. Additional capabilities of the tool will allow extension of the default routine library by the programmer and will allow interfacing with other hardware synthesis and codesign tools. The tool will be implemented as a code generation target in Ptolemy.

4.7 CONTROL AND SIGNAL PROCESSING

In the early part of the project it concentrated on the computational aspects of signal processing systems, and thus focused on models of computation such as dataflow that are particularly well suited. Toward the end, the attention broadened to include control and sequential decision-making aspects of system design. We pursued three approaches for combining control-oriented computation with data-oriented computation: hierarchical concurrent finite-state machines, the synchronous/reactive model of computation, and dynamically evaluated higher-order functions. This work is ongoing, with the parts completed under this project being seminal. The following specific accomplishments are reported:

4.7.1 The FSM Domain

Signal processing systems perform intensive numeric computation, but they typically also have sophisticated control logic for sequencing the computation tasks, switching among operation modes, coordination, and configuration. Dataflow models are suitable for describing numeric computations. The finite-state machine (FSM) is an intuitive model for describing control logic with a formal, well-studied mathematical theory. But the basic FSM model, which is flat and sequential, is not suitable for describing complex concurrent control. A common solution to this problem is hierarchical FSMs, which extend the basic FSM model with hierarchy and concurrency. The Statecharts visual formalism is an example of this approach.

We observe that FSM semantics, hierarchy, and concurrency are orthogonal semantic properties of Statecharts. If we take away from Statecharts the transitions that cross hierarchy boundaries, we get a simpler model in which FSM semantics can be cleanly separated from concurrency semantics. This means that the basic FSM model can be mixed with the various concurrency models to get many models that are only slightly weaker than Statecharts. We call this new computational model "*charts," where the "*" is a wildcard representing various possible concurrency models.

We have created a preliminary implementation of this model in Ptolemy. Systems can be built by hierarchically nesting FSMs and concurrency models. The synchronous dataflow model is particularly attractive because when it is combined hierarchically with FSMs in certain ways, the combination is far more expressive than either SDF or FSMs alone, even though the resulting system remains finite state. Verification, synthesis, and optimization questions all remain decidable. We have developed a preliminary visual editor for state transition diagrams, which is integrated into the Ptolemy GUI so that a user can seamlessly traverse a hierarchical design that combines FSMs with dataflow block diagrams. At present we can simulate such a mixed-model system description. We plan to add the capability to generate code from such systems. Preliminary results are reported in [2].

4.7.2 Synchronous/Reactive Modeling

The synchronous/reactive model of computation is popular (mostly in Europe) for the design of real-time embedded systems. Examples of languages that use this model are Esterel, Lustre, Signal, and Argos. A key property of the model is that events in concurrent modules are totally ordered with respect to one another. This means that any two events are either simultaneous, or one unambiguously precedes the other. This contrasts the dataflow approach, where events are partially ordered. A second key property of SR languages is that simultaneous events are defined by a fixed-point equation. Fixed-point theory guarantees the existence of a least fixed point under certain technical conditions.

Stephen Edwards completed his Ph.D. thesis [74], which describes a coordination language that combines the synchronous/reactive model with the ability to assemble systems from heterogeneous pieces (i.e., described in a variety of languages). It presents a mathematical framework for dealing with zero-delay-induced paradoxes and presents a way to schedule systems with feedback. The abstract of the thesis effectively summarizes the results:

> "The need for new languages and paradigms for designing software for embedded computing systems continues to grow as general-purpose microcontrollers become faster and cheaper. Many of these systems need precise control over when things happen, yet few languages provide this facility. Another major challenge is handling the growing complexity of these systems.
>
> In this dissertation, I present a new model of computation for embedded system software that is the first to fuse precise control over timing with the ability to build systems from heterogeneous pieces. It combines the synchronous model of time (used in languages such as Esterel) with the hierarchical heterogeneity of the Ptolemy system. Heterogeneity addresses the complexity problem by allowing each subsystem to be designed using the best language.
>
> My two major contributions are the formal semantics of this model and an efficient, predictable execution scheme for it. Dealing with zero-delay feedback loops, a side-effect of the zero-delay assumption needed for synchrony, is the semantic challenge, and I solve it with a fixed-point scheme that guarantees all systems are deterministic by construction. The execution scheme I present is provably correct and eliminates run-time scheduling overhead by making all decisions before the system is run.
>
> I present results that show my model of computation is both efficient and can be used to implement practical systems. It is my hope that these ideas will be used in the future to make designing complex time-critical embedded software easier and less error-prone."

SR languages have been used in control-intensive, safety-critical embedded system designs such as aircraft and nuclear power-plant control. Their formal properties ensure determinacy and bounded memory, and enable extensive verification. They appear to be an attractive model for certain kinds of signal processing systems.

Stephen constructed an SR domain in Ptolemy that differs from standard SR languages by allowing modules to be designed in some foreign model of computation. This is consistent with the "hierarchical heterogeneity" principle of Ptolemy. This domain has a number of practical and theoretical challenges that result from this heterogeneity. In particular, the information-hiding principle used in Ptolemy occludes certain important information about modules that is normally exploited in compiling these languages. We have had to adapt the theory and compilation techniques to avoid violating this information hiding.

Stephen developed a dynamic execution policy for the SR domain in Ptolemy and proved that it always converges to the minimal fixed point. We have a theoretical bound on the number of steps required to reach this fixed point (order N^2, where N is the number of actors in the graph) and have been developing heuristics that fall well below this bound.

The first nontrivial application of the synchronous/reactive (SR) domain involved the interpretation of MIDI control signals to control sound synthesis. This application provided a suitable representative of systems that combine intensive signal processing with intensive control logic. A Midi keyboard interfaced to the serial port of a Sparc 10 provided the control sequence. The Midi keyboard was capable of providing highly complex and time-sensitive control signals, thus representing a demanding system environment. The Synchronous Reactive domain and SDF (synchronous dataflow) domains were used to create a synthesizer using the Sparc 10 to generate sound, and the keyboard to trigger events.

4.7.3 Dynamically Evaluated Higher-Order Functions

We prototyped C++ and Tcl interfaces to the dynamic higher-order functions mechanism, in which we dynamically switch in a replacement block. This can be used to implement hierarchical state machines (with no cross-hierarchy state transitions), and dynamically evaluated higher-order functions. For example, we can implement conditionals (like if-then-else) within a dataflow actor as a HOF by using the C++ interface.

4.7.4 Open Problems

Demonstration systems have been constructed where modules written in the synchronous/reactive domain are embedded within both discrete event (DE) and synchronous dataflow (SDF) systems. We have observed that while the use of SDF in this context may be adequate for hardware design, it has serious inefficiencies for embedded software design. Moreover, the problems are fundamental to the embedding of any technique where events are totally ordered (as in SR) within dataflow graphs, where events are only partially ordered.

The nature of the problem is as follows: to preserve determinacy, the dataflow model does not permit actors to test their input ports for the presence of a token, nor to take a branch depending on whether a token is present. However, a controller often wants to monitor a signal, say an exception signal, and branch in response to that signal. In the SDF embedding, that signal must always be present, using for example a Boolean FALSE to indicate that an exception has not occurred, and a Boolean TRUE to indicate that an exception has occurred. For circuit design, where this signal may represent a voltage on a wire, there is no inefficiency implied here. For software, however, the production and consumption of a large number of FALSE tokens that indicate that nothing interesting is happening can be quite costly. It is an ongoing effort in our group to attach reasonable semantics to this sort of combination.

4.8 FORMAL METHODS

The focus of the work on formal methods was to understand models of computation that can be applied to system-level design of embedded signal processing systems. The major focus, therefore, was on concurrent models and models that coexist well with huge computational loads and real-time constraints.

4.8.1 A Semantic Framework for Comparing Models of Computation

In collaboration with Professor Alberto Sangiovanni-Vincentelli, we developed a denotational framework (a “meta model”) within which certain properties of models of computation can be understood and compared. It describes concurrent processes in general terms as sets of possible behaviors. Compositions of processes are given as intersections of their behaviors. The interaction between processes is through signals, which are collections of events. A system is determinate if, given the constraints imposed by the inputs, there are exactly one or exactly zero behaviors. Each event is a value-tag pair, where the tags can come from a partially ordered or totally ordered set. Timed models are where the set of tags is totally ordered. Synchronous events share the same tag, and synchronous signals contain events with the same set of tags. Synchronous systems contain synchronous signals. Strict causality (in timed systems) and continuity (in untimed systems) ensure determinacy under certain technical conditions. The framework is used to compare certain essential features of various models of computation, including Kahn process networks, dataflow, sequential processes, concurrent sequential processes with rendezvous, Petri nets, concrete data structures, and discrete-event systems. Details are reported in [44][66].

4.8.2 Semantics of Discrete-Event Systems

In what could be a significant breakthrough, we followed up on a suggestion by Gerard Berry of INRIA to develop a semantic model of discrete-event systems (such as that used in VHDL, Verilog, and the discrete-event domain in Ptolemy). This model provides a complete metric space for signals in such systems, thereby enabling the use of standard, well-established mathematical methods (most notably the Banach fixed point theorem) to study issues such as determinacy. This work is included in [66].

4.8.3 Semantics of Dataflow

We formally characterized the previously informal relationship between dataflow and Kahn process networks. In KPNs, a "process" is a functional mapping from input sequences to output sequences, where the function is constrained to be continuous in a complete partial order (CPO). The CPO is based on the so-called "prefix order." Dataflow is a special case where the process is constructed as a sequence of "firings" where a firing is an atomic quantum of computation. The difficulty in the past has been in formally defining the constraints on the firing function and the firing rules (which indicate when a firing can occur) such that the resulting process is continuous. Continuity is desirable because it ensures determinacy.

The firing rules can be characterized as a set of signals (tuples of sequences of tokens). When one of the firing rules is a prefix of the input to the process, then the firing function "fires," consuming the prefix and producing output tokens. The process is then recursively applied to the remaining input tokens. If no two members of the firing rule set are "joinable" (meaning that they have an upper bound in the CPO), and the firing function is continuous, then the process will be continuous. A key part of the result is showing that the recursive definition of the process in terms of the firing functions is sensible and determinate. This can be done by defining a CPO on functions and showing that the process can be given in terms of the firing function by a continuous functional (which maps functions into functions).

These results are reported in [67].

4.8.4 Dataflow and Functional Languages

In [14], we reviewed a mathematical theory of dataflow based on partial orders, and connect this theory to the functional languages and dataflow architectures communities. A central idea is that a dataflow process consists of repeated applications of dataflow firings, and that this can be described by the higher-order function $F = \text{map}(f)$, where f is a function describing a single actor firing. The "map" higher-order function applies f to a stream input. This notation formalizes a number of concepts that have not been clear (at least not to us). We have determined, for example, that if "f" is "sequential" (in a very technical sense), then "F" is sequential. Sequentiality implies determinacy of a network of such functions. The next broader class of functions that we know of beyond the sequential functions, called "stable functions," also imply determinacy. However, we have found a counterexample where F is not stable even though f is. For this counterexample, F is not determinate. Thus, we believe that sequential functions characterize, in a very fundamental sense, those functions whose composition abstracts to a determinate function. The class of sequential functions, as it happens, is exactly the class implemented by the Ptolemy Dynamic Dataflow (DDF) domain.

5. Software

The Ptolemy software serves as both a laboratory for experimentation and a mechanism for disseminating results. During the course of the project, we completed three major software releases and several minor ones. The major enhancements of each release are summarized below. Version numbers begin with "0" to emphasize that this is research software, not a commercial product.

5.1 INFORMATION DISSEMINATION POLICY

We set up a Web site, http://ptolemy.eecs.berkeley.edu, that was used to distribute all software (including source code) and documentation (in PostScript, HTML and PDF, together with updated summary sheets, answers to frequently asked questions, a quick tour, and a tutorial). We set up a Usenet news group called comp.soft-sys.ptolemy and a mailing list ptolemy-hackers@ptolemy.eecs.berkeley.edu. Postings to the mailing list are cross-posted to the news group. Postings are archived and searchable from our World Wide Web site.

5.2 PTOLEMY 0.5 (FEBRUARY 1994)

5.2.1 Major New Features

Major features introduced in the 0.5 version include:

- Greatly improved documentation (see below).
- Extensible, animated, interactive GUI based on Tcl/Tk.
- The boolean dataflow domain.
- VHDL code generation.
- Silage code generation.
- Fast discrete-event scheduling.
- A communicating processes domain for event-driven simulation of hardware systems.
- Fixed-point simulation.
- Matrix data types and functional blocks.

5.2.2 Documentation

With the objective of making Ptolemy more usable both within Berkeley and outside, we completely rewrote the documentation. The 0.4.1 version had been written using troff. The 0.5 version was converted to use FrameMaker, and used a more tutorial, more narrative style with extensive use of graphics. The complete manual, called "The Almagest," is divided into four volumes:

- The User's Manual
- The Star Atlas
- The Programmer's Manual
- The Kernel Manual

The first two are intended for users who will not be writing code to extend the system. The third is for users who will be writing new functional blocks (called stars), and the fourth is for users who will be extending the system in more fundamental ways, such as by adding new models of computation or new synthesis tools.

The User's manual and Kernel manual have both been converted to HTML for on-line, hypertext access. Also providing improved on-line documentation, two self-guided tours of Ptolemy are distributed with the system:

- A "Quick Tour" takes the user through the features of the more mature Ptolemy domains.
- A "What's New" tour guides the user through an overview of what has been added in each new version of Ptolemy.

5.3 PTINY 0.5 (APRIL 1994)

The "Ptiny" release is a demonstration subset that is easy to install and requires much less disk space than the full system. A number of our regular users started with this version. Moreover, this version is designed to fully support our instructional uses of Ptolemy.

5.4 PTOLEMY 0.5.1 (SEPTEMBER 1994)

5.4.1 Major New Features

Major features introduced in the 0.5.1 version include:

- A Matlab interface.
- Higher-order functions.

- Multidimensional synchronous dataflow.
- Initializable delays.

5.5 PTOLEMY 0.5.2 (MAY 1995)

The Ptolemy 0.5.2 release, which consist of approximately 2000 files containing 300,000 lines and 8 Mb of source code, was distributed in May of 1995.

5.5.1 Major New Features

This was an incremental release containing three major features:

- Greatly enhanced simulation speed,
- A library of interactive graphical widgets, and
- Support for higher-order functions in all domains.

5.5.2 Platforms

Internally developed: Sun Sparc (SunOS and Solaris), HP (HP/UX), and SGI (Irix).

Contributed by outside users: Dec Alpha (Ultrix), PC (Linux), IBM RS/6000 (AIX), and Power PC (AIX).

Additional external ports: DecStation (Ultrix) and PC (NetBSD).

5.6 PTOLEMY 0.6 (APRIL 1996)

The Ptolemy 0.6 release consists of approximately 3000 files containing 400,000 lines and 9 Mb of source code (compressed).

5.6.1 Domains

- Multidimensional synchronous dataflow, MDSDF.
- New functional VHDL domain.
- Process networks, PN.
- Extension of Boolean dataflow (BDF) to integer-controlled dataflow (IDF).

5.6.2 Schedulers

- Loop scheduler.
- Dynamic dataflow scheduler that maintains bounded memory.

- Partitioning SDF applications into multiple VHDL hardware modules.
- Buffer-optimal loop scheduler for acyclic SDF graphs.
- Latency constrained resynchronization (LCR) algorithm for 2 processor systems that is capable of handling delays.

5.6.3 Automatic Code Generation

- Synthesis of parallel code for network of workstations (NOW).
- Multi-lingual code generation (C-based wormholes).
- Tunable VHDL code generation (sequential or synthesizable).
- Incrementally compiled code generation subsystems.

5.6.4 Visualization

- Interfaces to a freely available graph visualization program: printDot - outputs a galaxy hierarchy in dotty format, printClusterDot - outputs the galaxy in it's clustered form.
- Tcl/Tk versions of Gantt chart and logic analyzer.
- Interface to Tycho (see below).

5.6.5 Ptolemy Infrastructure

- Tcl parameter expression parser.
- Ptolemy Makefile redesign.
- Matlab Tcl interface.
- Tcl/Mathematica interface.
- First-cut at design-methodology management.
- Code generation wormholes.
- Portable scheduler file format.
- Heterogeneous code generation (VHDL, 56K, C).
- C++ documentation generation system.
- HTML documentation of Ptolemy stars.
- File datatype.
- Automatic generation of Ptcl scripts from block diagrams.
- New iterator classes for various kernel classes.
- Itcl incorporation, an object-oriented extension to Tcl/Tk.
- Script for creating custom versions of Ptolemy.

5.6.6 Platforms

Platforms that we distribute binaries for: Solaris2.4, HPUX-10.01, SunOS4.1.3.

Platforms that Ptolemy 0.6 has been compiled for: IBM AIX3.2.5, DEC Alpha OSF/1 V3.2, FreeBSD 2.1-Stable, Irix5.3, Irix6.x, HPUX9.x, Linux Slackware3.0, Solaris2.5, HPUX10.01, HP CC.

5.7 PTOLEMY 0.7 (JUNE 1997)

Ptolemy 0.7 and Tycho 0.2 were released on June 13th, 1997.

5.7.1 Domains

- FSM — finite state machines
- SR — synchronous/reactive.

5.7.2 Schedulers

- An optimized acyclic loop scheduler. This scheduler does joint code/data minimization; it generates single appearance schedules optimized for buffer memory usage. It is useful in code generation, especially in assembly language code generation for embedded signal processors which have limited program and data memory.

5.7.3 Code Generation

- Code generation for the TI C50 DSP, CGC50.
- UltraSparc VIS (visual instruction set) code generation.
- Real-time CD-quality audio on Ultrasparc workstations.
- Synthesis of C code that is dynamically loaded into Tycho.
- Improved user interfaces (based on Tycho) for synthesized C programs.
- Visualization of implementation costs.

5.7.4 Visualization

- Integrated HTML documentation of functional blocks.
- Tycho 0.2 (see below).

5.7.5 Ptolemy Infrastructure

- A revamped type system.
- Retargeting tool.
- Scripted higher-order functions.

5.7.6 Platforms

We distribute binaries for Solaris2.5.1, Solaris 2.4, HPUX10.20, HPUX9.x, SunOS4.1.3, and DEC Alpha OSF1. Ptolemy has been compiled for a number of other platforms by users outside Berkeley.

5.7.7 Documentation

The User's Manual is about 500 pages with about 440 figures, tables and equations. The User's

Manual is available in HTML, PDF and PostScript from the Ptolemy Web page. In addition, there is a Programmer's Manual and Kernel Manual.

5.8 TYCHO

Tycho is an object-oriented syntax manager with an underlying heterogeneous technical rationale. It provides a number of editors and graphical widgets in an extensible, reusable framework. The editors for textual syntaxes are modeled after emacs in the sense the emacs key bindings are used whenever possible. However, they make more extensive use of menus, windows, and dialogs than emacs. Also, the intent is that visual editors and visualization tools will be fully integrated, something that would be difficult to accomplish with emacs in its current form. Editors for visual syntaxes will be more diverse. The system documentation is integrated, using a hypertext system compatible with the World Wide Web.

Tycho was originally conceived for use with Ptolemy system, but it has grown into a system that is useful on its own. Tycho has been used extensively in the development of the Tycho software itself.

Tycho is written primarily in Itcl, also called [incr Tcl], developed by Michael McLennan of AT&T. Itcl is an object-oriented extension of Tcl, a "tool command language" written by John Ousterhout of U.C. Berkeley, now under continued development at Sun Microsystems. The window toolkit Tk and its object-oriented extension Itk are also used extensively.

5.8.1 Objectives

- To build a genuinely object-oriented user interface, where multiple visual syntaxes can be combined, and application-specific visual syntaxes can be constructed.
- To provide an extensible framework for experimentation with visual syntaxes, where mundane tasks such as documentation, font management, color management, and dialogs with the user are built using a shared, common infrastructure.
- To extend the non dogmatic nature of the Ptolemy kernel (which supports multiple semantic models) to the user interface (which will support multiple syntactic models).
- To experiment with design visualization, broadening the perspective beyond a schematic or block-

diagram perspective of Ptolemy, and exploring new visual and mixed visual/textual syntaxes for design representation and understanding.

- To leverage off work in the Tcl/Tk community to get portable (Unix, Microsoft Windows PCs and Apple Macintoshes) code.
- To design a sophisticated, extensible, interactive documentation system.

One of the key principles in Tycho is that anything can have a hyperlink to anything else. Documentation will have links to source code, and vice versa. Visual editors will have links to textual editors. And specialized displays can be created for any form of data. These displays, of course, are also connected by hyperlinks.

An interim mechanism is provided where Tycho forms a subsystem within the much older visual editor for Ptolemy called "pigi" (which stands for Ptolemy interactive graphical interface).

5.8.2 Tycho 0.1 Release (March 1996)

Tycho 0.1 was released with Ptolemy 0.6. It was still a very preliminary system. It included:

- Visual editors and displays for various types of graphs.
- Syntax-sensitive text editors for Itcl, HTML, C, C++, Ptlang (Ptolemy star) files, Java and Esterel.
- Interactive shells communicating with Tcl, Matlab, Mathematica.
- Graphical editors
- Integrated, HTML-based documentation.
- Indexes and index browsers.
- A family of dialog windows.
- Context sensitive spell checker.
- Font and color management system.
- Error handling with a stack display.
- Auto-save.
- Some elementary data structures: Stack, CircularList, Graph, DirectedAcyclicGraph, Forest.

5.8.3 Tycho 0.1.1 Release (December 1996)

Tycho 0.1.1 was released on December 17, 1996. This was an interim release that improved performance and added many new features. Most notably:

- View/Displayer architecture.
- Slate object for managing composite graphical objects.

- Popup menus (to complement pre-existing pull-down menus)
- Menu-bar object.
- Glimpse Index Browser.
- Graphical Itcl Class browser
- Preferences manager
- Exec class which can be used to run remote programs like make.
- Print dialog.
- Java syntax sensitive editor.
- Windows NT port.
- Hierarchical indexes.

5.8.4 Tycho 0.2 Release (June 1997)

Tycho 0.2 was released with Ptolemy 0.7. Significant new features:

- Java/Tycho interface
- Compilation and dynamic loading of C modules at runtime.
- Improved preferences manager.
- An interface to C,C++ and Java compilers.
- Interfaces to SCCS and RCS revision control systems.
- An interface to the Glimpse index browser, which can rapidly search large directory trees.
- A graphical Tcl profiler.
- A source code documentation system and browser.
- A Tycho Information Model (TIM) architecture.
- A time-slice scheduler for dynamically linked C modules.

5.9 TMATH

Ptolemy 0.7 comes with an interface to Matlab 4.2 and Mathematica 2.2, but Ptolemy must be recompiled for a user to access the interface. Since the 0.7 release, Brian Evans (UT Austin) has upgraded the interface to be compatible with Matlab 4.2 and 5.0 as well as Mathematica 2.2 and 3.0, which will likely be released in the next version of Ptolemy and as a patch to the current version. In the meantime, Brian has spun off the interface to Matlab and Mathematica from C++ and Tcl as a separate tool called TMath (version 0.2).

The TMath package is an extension to Tcl that allows Tcl 7.x to control Matlab and Mathematica processes and to evaluate Matlab and Mathematica commands, either through scripts or interactive sessions. It works with Matlab 4.2 and 5.0 as well as Mathematica 2.2 and 3.0. TMath provides:

- two new Tcl commands matlab and mathematica,
- a framework for registering Tcl commands implemented as C++ methods,
- C++ interfaces for Matlab and Mathematica, and

C++ objects to control multiple Matlab and Mathematica processes. TMath will work on all of the architectures supported by the Ptolemy software environment. For more information about TMath, see

http://www.ece.utexas.edu/~bevans/projects/tmath.html

6. Acknowledgments

6.1 PARTICIPANTS AT BERKELEY

6.1.1 Principal Investigator

- Edward A. Lee

6.1.2 Professional Staff

- Diane Chang
- Kevin Chang
- Christopher Hylands
- Alan Kamas
- Mary Stewart

6.1.3 Post-Doctoral Researchers

- Brian L. Evans
- Alain Girault
- Seehyun Kim
- Praveen Murthy
- Rajagopal Nagarajan
- John Reekie
- Dick Stevens (from NRL)
- Juergen Teich

6.1.4 Graduate Students

- Shuvra S. Bhattacharyya
- Wan-Teh Chang
- Michael Chen
- William Chen
- Cliff Cordeiro
- John Davis, II
- Stephen Edwards
- Ron Galicia
- Mudit Goel
- Michael Goodwin
- Sangjin Hong
- Asawaree Kalavade
- Joel King
- Allen Lao
- Bilung Lee

- William Li
- Jie Liu
- Praveen K. Murthy
- Thomas M. Parks
- José Luis Pino
- Farhana Sheikh
- Sun-Inn Shih
- Neil Smyth
- S. Sriram
- Patrick J. Warner
- Michael C. Williamson
- Mei Xiao
- Yuhong Xiong

6.1.5 Undergraduate Students

- Raza Ahmed
- Sunil Bhave
- Kang Ngee Chia
- Steve X. Gu
- Luis Gutierrez
- Farhad Jalilvand
- Yu Kee Lim
- Siamak Modjtahedi
- Matthew Tavis
- William Tsu

6.2 PARTICIPANTS OUTSIDE BERKELEY

We would like to give special thanks to the following people outside our group who contributed in significant ways to the results reported here:

- Egbert Ammicht (AT&T)
- Neal Becker (Comsat)
- Shuvra S. Bhattacharyya (Hitachi)
- Joseph T. Buck (Synopsys)
- Gyorgy Csertan (Technical University of Budapest)
- Dan Ellis (MIT Laboratory for Computer Science)
- Pravil Gupta (Compass Design Automation)
- Soonhoi Ha (Seoul National University, Korea)
- Alexander Kurpiers (T.U. Darmstadt, Germany)
- Tom Lane (Structured Software Systems)

- Douglas Niehaus (Univ. of Kansas)
- Eric Pauer (Lockheed Sanders)
- Sunil Samel (IMEC, Belgium)
- Christopher Scannell (NRL)
- Richard Tobias (White Eagle Systems Technology)
- Stefan De Troch (IMEC)
- Tom Truman (U.C. Berkeley)
- Alberto Vignani (Fiat) Xavier Warzee (Thomson CSF)
- Joseph M. Winograd (Boston University)
- Fritz Heinrichmeyer (FernUniverstitat in Hagen, Germany)
- Uwe Trautwein (Technical University of Ilmenau, Germany)
- Xavier Warzee (Thomson CSF)

6.3 CORPORATE SUPPORT

6.3.1 Sponsors

The following organizations have contributed additional financial support for the Ptolemy project:

- the State of California MICRO program
- The Alta Group of Cadence Design Systems
- Dolby Laboratories
- Hitachi
- LG Electronics
- Lockheed Martin ATL
- Motorola
- NEC
- Philips
- Rockwell
- the Semiconductor Research Corporation (SRC)

6.3.2 Assistance With Software

The following organizations have helped, particularly by evaluating alpha releases and contributing ports to platforms that we do not have at Berkeley:

- Ericsson
- HDA
- Hewlett-Packard
- Lincoln Labs
- Meta-Software
- MIT Lincoln Labs

- Ncube
- Structured Software Systems
- Technische Universitaet Ilmenau, Germany
- Thomson CSF
- Thomson Multimedia
- Univ. of Maryland Baltimore County
- Univ. of Texas, Austin
- White Eagle Systems Technology, Inc.

7. Publications

7.1 BOOKS AND CHAPTERS

[1] S. S. Bhattacharyya, P. K. Murthy and E. A. Lee, *Software Synthesis from Dataflow Graphs*, Kluwer Academic Publishers, Norwell, Mass, 1996.

[2] W.-T. Chang, A. Kalavade, and E. A. Lee, "Effective Heterogeneous Design and Cosimulation," chapter in *Hardware/Software Co-design*, G. DeMicheli and M. Sami, eds., NATO ASI Series Vol. 310, Kluwer Academic Publishers, 1996. Also presented at NATO Advanced Study Institute Workshop on Hardware/Software Codesign, Lake Como, Italy, June 18 — 30, 1995. (http://ptolemy.eecs.berkeley.edu/papers/effective)

7.2 JOURNAL ARTICLES

[3] S. S. Bhattacharyya, J. T. Buck, S. Ha, and E. A. Lee, "Generating Compact Code from Dataflow Specifications of Multirate Signal Processing Algorithms," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol. 42, no. 3, pp. 138-150, March 1995.

[4] S. S. Bhattacharyya and E. A. Lee, "Memory Management for Dataflow Programming of Multirate Signal Processing Algorithms," *IEEE Trans. on Signal Processing*, vol. 42, no. 5, May 1994. (http://ptolemy.eecs.berkeley.edu/papers/buffering.ps.Z)

[5] S. S. Bhattacharyya and E. A. Lee, "Looped Schedules for Dataflow Descriptions of Multirate Signal Processing Algorithms," *Formal Methods in System Design*, No. 5, No. 3, December, 1994. (updated from Technical Report, May 21, 1993). (http://ptolemy.eecs.berkeley.edu/papers/loop_schedules.ps.Z)

[6] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "APGAN and RPMC: Complimentary Heuristics for Translating DSP Block Diagrams into Efficient Software Implementations," *Journal of Design Automation for Embedded Systems*, Vol. 2, No. 1, pp. 33-60, January, 1997. (http://ptolemy.eecs.berkeley.edu/papers/96/daem-apgan)

[7] S. S. Bhattacharyya, S. Sriram, and E.A. Lee, "Optimizing Synchronization in Multiprocessor DSP Systems," *IEEE Tr. on Signal Processing*, Vol. 45, No. 6, June 1997. (http://ptolemy.eecs.berkeley.edu/papers/97/synchronization/)

[8] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," *Int. Journal of Computer Simulation*, special issue on "Simulation Software Development," vol. 4, pp. 155-182, April, 1994. (http://ptolemy.eecs.berkeley.edu/papers/JEurSim).

[9] W.-T. Chang, S.-H. Ha, and E. A. Lee, "Heterogeneous Simulation — Mixing Discrete-Event Models with Dataflow," RASSP special issue of the *Journal on VLSI Signal Processing*, Vol. 13, No. 1, January, 1997. (http://ptolemy.eecs.berkeley.edu/papers/96/heterogeneity)

[10] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis," *Proceedings of the IEEE*, Vol. 85, No. 3, March 1997. (http://ptolemy.eecs.berkeley.edu/papers/97/codesign)

[11] S. Ha and E. A. Lee, "Compile-Time Scheduling of Dynamic Constructs in Dataflow Program Graphs," *IEEE Trans. on Computers*, Vol. 46, No. 7, July 1997.

[12] A. Kalavade and E. A. Lee, "Complexity Management in System-Level Design," *Journal of VLSI Signal Processing Systems*, Vol. 14, No. 2, November 1996. (http://ptolemy.eecs.berkeley.edu/papers/96/jvlsi-dmm)

[13] A. Kalavade and E. A. Lee, "The Extended Partitioning Problem: Hardware/Software Mapping and Implementation-Bin Selection," *Journal of Design Automation for Embedded Systems*, vol. 2, March 1997, pp. 125-163. (http://ptolemy.eecs.berkeley.edu/papers/daem-partitioning-95)

[14] E. A. Lee and T. M. Parks, "Dataflow Process Networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773-801, May, 1995. (http://ptolemy.eecs.berkeley.edu/papers/processNets)

[15] P. K. Murthy, S. S. Bhattacharyya, and E. A. Lee, "Combined Code and Data Minimization for Synchronous Dataflow Programs," *Journal of Formal Methods in System Design*, July 1997 (http://ptolemy.eecs.berkeley.edu/papers/jointCodeDataMinimize)

[16] J. L. Pino, S. Ha, E. A. Lee, and J. T. Buck, "Software Synthesis for DSP Using Ptolemy," *Journal on VLSI Signal Processing*, vol. 9, no. 1, pp. 7-21, Jan., 1995. (http://ptolemy.eecs.berkeley.edu/papers/jvsp_codegen)

[17] S. Sriram and E. A. Lee, "Determining the Order of Processor Transactions in Statically Scheduled Multiprocessors," *Journal of VLSI Signal Processing*, Vol. 15, No. 3, pp. 207-220, March 1997. (http://ptolemy.eecs.berkeley.edu/papers/97/order)

7.3 CONFERENCE PAPERS

[18] G. Arslan, B. L. Evans, F. A. Sakarya, and J. L. Pino, "Performance Evaluation and Real-Time Implementation of Subspace, Adaptive, and DFT Algorithms for Multi-Tone Detection," *Proc. Int. Conf. on Telecommunications*, Istanbul, Turkey, April 15-17, 1996. (http://ptolemy.eecs.berkeley.edu/papers/96/dtmf_ict)

[19] R. H. Bamberger, B. L. Evans, E. A. Lee, J. H. McClellan, and M. A. Yoder, "Integrating Layout, Analysis, and Simulation Tools in Electronic Courseware for Teaching Signal Processing," Invited Paper, *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, May 8-12, 1995, Detroit, MI, pp. 2873-2876. (http://ptolemy.eecs.berkeley.edu/papers/education)

[20] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Converting Graphical DSP Programs into Memory-Constrained Software Prototypes," *Proc. of IEEE Int. Workshop on Rapid Systems Prototyping*, Chapel Hill, NC, June 7-9, 1995. (http://ptolemy.eecs.berkeley.edu/papers/PganRpmcDppo/)

[21] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Optimal Parenthesization of Lexical Orderings for DSP Block Diagrams," *Proc. IEEE Workshop on VLSI Signal Processing*, Osaka, Japan, October 16-18, 1995. (http://ptolemy.eecs.berkeley.edu/papers/PganRpmcDppo/)

[22] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Software Synthesis for Synchronous Dataflow," *Proc. International Conference on Application Specific Systems, Architectures, and Processors*, July, 1997, invited paper.
(http://ptolemy.eecs.berkeley.edu/papers/97/software)

[23] S. S. Bhattacharyya, S. Sriram, and E. A. Lee, "Minimizing Synchronization Overhead in Statically Scheduled Multiprocessor Systems," *Proc. of IEEE Int. Conference on Application Specific Array Processors*, July 24-26, 1995.
(http://ptolemy.eecs.berkeley.edu/papers/synchOpt/)

[24] S. S. Bhattacharyya, S. Sriram, and E. A. Lee, "Latency-Constrained Resynchronization for Multiprocessor DSP Implementation," *Proc. ASAP Conference,* Chicago, August 19-21, 1996.
(http://ptolemy.eecs.berkeley.edu/papers/96/resync)

[25] S. S. Bhattacharyya, S. Sriram, and E. A. Lee, "Self-Timed Resynchronization: A Post-Optimization for Static Multiprocessor Schedules," *Proc. IPPS,* April 1996.
(http://ptolemy.eecs.berkeley.edu/papers/96/parallel)

[26] J. T. Buck, "A Dynamic Dataflow Model Suitable for Efficient Mixed Hardware and Software Implementations of DSP Applications," *Proc. of Codes/CASHE 94, Third International Workshop on Hardware/Software Codesign*, Grenoble, France, Sept. 22-24, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/DDF_codesign.ps.Z)

[27] M. J. Chen and E. A. Lee, "Design and Implementation of a Multidimensional Synchronous Dataflow Environment," Invited Paper, *Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 31 - Nov. 2, Pacific Grove, CA, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/mdsdf_asilomar.ps.Z)

[28] W. Chen, H. J. Reekie, S. Bhave, and E. A. Lee, "Native Signal Processing on the UltraSparc in the Ptolemy Environment," *Proc. of the 30th Annual Asilomar Conference on Signals, Systems, and Computers*, November 1996.
(http://ptolemy.eecs.berkeley.edu/papers/96/ultrasparc/)

[29] K. Chiang, B. L. Evans, W. T. Huang, F. Covet, E. A. Lee, H. J. Reeky, D. G. Messerschmitt, and S. Sister, "Real-Time DSP for Sophomores," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Atlanta, GA, May 7-10, 1996, vol. 2, pp. 1097-1100.
(http://ptolemy.eecs.berkeley.edu/papers/96/realTimeCourse)

[30] B. L. Evans, D. R. Firth, K. D. White, and E. A. Lee, "Automatic Generation of Programs That Jointly Optimize Characteristics of Analog Filter Designs," *Proc. of European Conference on Circuit Theory and Design*, August 27-31, 1995, Istanbul, Turkey, pp. 1047-1050. (http://ptolemy.eecs.berkeley.edu/papers/filter_design_ecctd95.ps.Z)

[31] B. L. Evans, S. X. Gu, A. Kalavade, and E. A. Lee, "Symbolic Computation in System Simulation and Design," Invited Paper, *Proc. of SPIE Int. Sym. on Advanced Signal Processing Algorithms, Architectures, and Implementations*, July 9-16, 1995, San Diego, CA, pp. 396-407.
(http://ptolemy.eecs.berkeley.edu/papers/spie95symbcomp.ps.Z)

[32] B. L. Evans, S. X. Gu, and R. H. Bamberger, "Interactive Solution Sets as Components of Fully Electronic Signals and Systems Courseware," *Proc. of IEEE Asilomar Conf. on Signals, Systems,*

and Computers, Oct. 31 - Nov. 2, Pacific Grove, CA, 1994, pp. 1314-1319.
(http://ptolemy.eecs.berkeley.edu/papers/education)

[33] B. L. Evans, A. Kamas, and E. A. Lee, "Design and Simulation of Heterogeneous Systems Using Ptolemy," *First Annual Rapid Prototyping of Application Specific Signal Processors (RASSP) Conference*, Arlington, VA, Aug. 15-18, 1994, pp. 97-105.
(http://ptolemy.eecs.berkeley.edu/papers/heterogeneous)

[34] B. L. Evans and J. H. McClellan, "Algorithms for Symbolic Linear Convolution," *Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 31 - Nov. 2, Pacific Grove, CA, 1994, pp. 948-953.
(http://ptolemy.eecs.berkeley.edu/papers/symbolic_convolution)

[35] B. L. Evans, J. Teich, and C. Schwarz, "Automated Design of Two-Dimensional Rational Decimation Systems," *Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 31 - Nov. 2, Pacific Grove, CA, 1994, pp. 498-502.
(http://ptolemy.eecs.berkeley.edu/papers/decimator_design)

[36] B. L. Evans, J. Teich, and T. A. Kalker, "Families of Smith Form Decomposition to Simplify Multidimensional Filter Bank Design," *Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 31 - Nov. 2, Pacific Grove, CA, 1994, pp. 363-367.
(http://ptolemy.eecs.berkeley.edu/papers/mD_filter_bank_design)

[37] C. Hylands, E. A. Lee, and H. J. Reekie, "The Tycho User Interface System," to be presented at the *5th Annual Tcl/Tk Conference* '97, Boston, Massachusetts, July, 1997.
(http://ptolemy.eecs.berkeley.edu/papers/97/tcltk-97/)

[38] A. Kalavade and E. A. Lee, "Manifestations of Heterogeneity in Hardware/Software Codesign," *Proc. of IEEE Design Automation Conference*, San Diego, CA, June, 1994, pp. 437-438
(http://ptolemy.eecs.berkeley.edu/papers/codesign)

[39] A. Kalavade and E. A. Lee, "A Global Criticality / Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem," *Proc. of Codes/CASHE 94, Third IEEE International Workshop on Hardware/Software Codesign*, Grenoble, France, Sept. 22-24, 1994, pp 42-48.
(http://ptolemy.eecs.berkeley.edu/papers/codesign)

[40] A. Kalavade and E. A. Lee, "The Extended Partitioning Problem: Hardware/Software Mapping and Implementation-Bin Selection," *Proc. of IEEE Int. Workshop on Rapid Systems Prototyping*, Chapel Hill, NC, June 7-9, 1995.
(http://ptolemy.eecs.berkeley.edu/papers/extended_partitioning/)

[41] A. Kalavade, J. L. Pino, and E. A. Lee, "Managing Complexity in Heterogeneous Specification, Simulation, and Synthesis," Invited Paper, *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, May 8-12, 1995, Detroit, MI, pp. 2833-2836.
(http://ptolemy.eecs.berkeley.edu/papers/managing_complexity)

[42] K. Khiar and E. A. Lee, "Modeling Radar Systems Using Hierarchical Dataflow," *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Detroit, MI, May 8-12, 1995, pp. 3259-3262.
(http://ptolemy.eecs.berkeley.edu/papers/Radarsimu.ps.Z)

[43] E. A. Lee, "Computing and Signal Processing: An Experimental Multidisciplinary Course", *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. VI, pp. 45-48, Adelaide, Australia, April, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/education)

[44] E. A. Lee, and A. Sangiovanni-Vincentelli, "Comparing Models of Computation," *Proc. of ICCAD*, San Jose, CA, Nov. 10-14, 1996.
(http://ptolemy.eecs.berkeley.edu/papers/96/comparing/)

[45] R. Mani, H. S. Nawab, J. M. Winograd, and B. L. Evans, "Integrated numeric and symbolic signal processing in a heterogeneous design environment," *Proc. SPIE Int. Sym. on Advanced Signal Processing Algorithms, Architectures, and Implementations,* Denver, CO, August 12-15, 1996.
(http://ptolemy.eecs.berkeley.edu/papers/96/ipusPtolemy/)

[46] P. K. Murthy, S. S. Bhattacharyya, and E. A. Lee, "Minimizing Memory Requirements For Chain-Structured Synchronous Dataflow Programs," *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. II, pp. 453-456, Adelaide, Australia, April, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/chainmemman.ps.Z)

[47] P. K. Murthy and E. A. Lee, "Optimal Blocking Factors for Blocked, Non-Overlapped Multiprocessor Schedules," Invited Paper, *Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 31 - Nov. 2, Pacific Grove, CA, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/blocFac.ps.Z)

[48] P. K. Murthy and E. A. Lee, "An Extension of Multidimensional Synchronous Dataflow to Handle Arbitrary Sampling Lattices," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Atlanta, GA, May 7-10, 1996, vol. 6, pp. 3306-3309.
(http://ptolemy.eecs.berkeley.edu/papers/genMdsdf)

[49] T. M. Parks and E. A. Lee, "Non Preemptive Real-Time Scheduling of Dataflow Systems," *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Detroit, MI, May 8-12, 1995, pp. 3235-3238.
(http://ptolemy.eecs.berkeley.edu/papers/real_time)

[50] T. M. Parks, J. L. Pino, and E. A. Lee, "A Comparison of Synchronous and Cyclo-Static Dataflow," *Proc. IEEE Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, October 29 - November 1, 1995.
(http://ptolemy.eecs.berkeley.edu/papers/csdfVSsdf)

[51] J. L. Pino, S. S. Bhattacharyya and E. A. Lee, "A Hierarchical Multiprocessor Scheduling System for DSP Applications," *Proc. IEEE Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, October 29 - November 1, 1995.
(http://ptolemy.eecs.berkeley.edu/papers/hierStaticSched-asilomar-95)

[52] J. L. Pino and E. A. Lee, "Hierarchical Static Scheduling of Dataflow Graphs onto Multiple Processors," *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Detroit, MI, May 8-12, 1995, pp. 2643-2646.
(http://ptolemy.eecs.berkeley.edu/papers/hierStaticSched)

[53] J. L. Pino, T. M. Parks, and E. A. Lee, "Automatic Code Generation for Heterogeneous Multiprocessors," *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. II, pp. 445-

448, Adelaide, Australia, April, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/autoMultiCodeGen)

[54] J. L. Pino, T. M. Parks and E. A. Lee, "Mapping Multiple Independent Synchronous Dataflow Graphs onto Heterogeneous Multiprocessors," *Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, Oct. 31 - Nov. 2, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/multiIndepGraph)

[55] J. L. Pino, M. C. Williamson, and E. A. Lee, "Interface Synthesis in Heterogeneous System-Level DSP Design Tools," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Atlanta, GA, May 7-10, 1996, vol. 2, pp. 1268-1271.
(http://ptolemy.eecs.berkeley.edu/papers/interfaceSynthesis)

[56] C. Schwarz, J. Teich, A. Vainshtein, E. Welzl, and B. L. Evans, "Minimal Enclosing Parallelogram with Application," *ACM Sym. on Computational Geometry*, June 5-7, 1995, Vancouver, Canada. (http://ptolemy.eecs.berkeley.edu/papers/decimator_design)

[57] S. Sriram and E. A. Lee, "Statically Scheduling Communication Resources in Multiprocessor DSP Architectures," Invited Paper, *Proc. of IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 31 - Nov. 2, Pacific Grove, CA, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/ordering_asilomar94.ps.Z)

[58] J. Teich, S. Sriram, L. Thiele, and M. Martin, "Performance Analysis of Mixed Asynchronous-Synchronous Systems," *Proc. of the IEEE Workshop on VLSI Signal Processing,* Oct. 26 - 28, 1994, pp. 103-112. Proceedings published as IEEE VLSI Signal Processing VII
(http://ptolemy.eecs.berkeley.edu/papers/wvsp94.ps.Z)

7.4 TECHNICAL REPORTS

[59] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Two Complementary Heuristics for Translating Graphical DSP Programs into Minimum Memory Software Implementations," Technical Report UCB/ERL M95/3, Electronics Research Laboratory, University of California, Berkeley, CA 94720, January 10, 1995.
(http://ptolemy.eecs.berkeley.edu/papers/PganRpmcDppo/)

[60] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Renesting Single Appearance Schedules to Minimize Buffer Memory," Technical Report UCB/ERL M95/43, Electronics Research Lab., UC Berkeley, CA 94720, April, 1 1995.
(http://ptolemy.eecs.berkeley.edu/papers/PganRpmcDppo/)

[61] S. S. Bhattacharyya, S. Sriram, and E. A. Lee, "Resynchronization for Embedded Multiprocessors," Technical Report UCB/ERL M95/70, University of California, Berkeley, CA 94720, September, 1995.
(http://ptolemy.eecs.berkeley.edu/papers/synchOpt/)

[62] S. S. Bhattacharyya, S. Sriram, and E. A. Lee, "Resynchronization of Multiprocessor Schedules: Part 1 -- Fundamental Concepts and Unbounded-latency Analysis," Technical Report UCB/ERL M96/55, Electronics Research Laboratory, U. C. Berkeley, October, 1996.
(http://ptolemy.eecs.berkeley.edu/papers/96/resync1/)

[63] S. S. Bhattacharyya, S. Sriram, and E. A. Lee, "Resynchronization of Multiprocessor Schedules: Part 2 -- Latency-constrained Resynchronization," Technical Report UCB/ERL M96/56, Electronics Research Laboratory, U. C. Berkeley, October, 1996.
(http://ptolemy.eecs.berkeley.edu/papers/96/resync2/)

[64] A. Lao, "Heterogeneous Cell-Relay Network Simulation and Performance Analysis with Ptolemy," Technical Report UCB/ERL M94/8, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, February 17, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/cell_relay.ps.Z)

[65] E. A. Lee, "Dataflow Process Networks," Electronics Research Laboratory Report, Tech. Report UCB/ERL M94/53, University of California, Berkeley, CA 94720, July 1994.
(http://ptolemy.eecs.berkeley.edu/papers/processNets)

[66] E. A. Lee and A. Sangiovanni-Vincentelli, "A Denotational Framework for Comparing Models of Computation," Technical Report UCB/ERL M97/11, University of California, Berkeley, CA 94720, January 30, 1997.
(http://ptolemy.eecs.berkeley.edu/papers/97/denotational/)

[67] E. A. Lee, "A Denotational Semantics for Dataflow with Firing," Technical Report UCB/ERL M97/3, Electronics Research Laboratory, U. C. Berkeley, January 1997.
(http://ptolemy.eecs.berkeley.edu/papers/97/dataflow/)

[68] T. Miyazaki, "Integer-Controlled Dataflow in Ptolemy," Technical Report UCB/ERL No. 97/21, University of California, Berkeley, CA 94720, March 19, 1997.

[69] P. K. Murthy, S. S. Bhattacharyya, and E. A. Lee, "Combined Code and Data Minimization for Synchronous Dataflow Programs," Technical Report UCB/ERL M94/93, Electronics Research Laboratory, University of California, Berkeley, CA 94720, November 29, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/jointCodeDataMinimize).

[70] P. K. Murthy and E. A. Lee, "On the Optimal Blocking Factor for Blocked, Non-Overlapped Schedules," Technical Report UCB/ERL 94/46, University of California, Berkeley, CA 94720, June 6, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/opt_blocking_factor.ps.Z)

[71] P. K. Murthy, and E. A. Lee, "A Generalization of Multidimensional Synchronous Dataflow to Handle Arbitrary Sampling Lattices," Technical Report UCB/ERL M95/59, Electronics Research Lab, UC, Berkeley, CA 94720, March 20, 1995.
(http://ptolemy.eecs.berkeley.edu/papers/genMdsdf/).

[72] J. L. Pino, S. S. Bhattacharyya and E. A. Lee, *A Hierarchical Multiprocessor Scheduling Framework for Synchronous Dataflow Graphs*, UCB/ERL M95/36, May 30, 1995.
(http://ptolemy.eecs.berkeley.edu/papers/erl-95-36)

7.5 PH.D. THESES

[73] S. S. Bhattacharyya, *Compiling Dataflow Programs for Digital Signal Processing,* Tech. Report UCB/ERL 94/52, Ph.D. Thesis, Dept. of EECS, University of California, Berkeley, CA 94720, July 12, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/compiling_dataflow.tar.Z)

[74] S. A. Edwards, *The Specification and Execution of Heterogeneous Synchronous Reactive Systems*, Ph.D. Thesis, Technical Report UCB/ERL M97/31, University of California, Berkeley, May 1997.
(http://ptolemy.eecs.berkeley.edu/papers/97/sedwardsThesis/).

[75] A. Kalavade, *System Level Codesign of Mixed Hardware-Software Systems,* Ph.D. Thesis, Technical Report UCB/ERL 95/88, Dept. of EECS, University of California, Berkeley, CA 94720, September, 1995.
(http://ptolemy.eecs.berkeley.edu/papers/kalavadeThesis)

[76] P. K. Murthy, *Scheduling Techniques for Synchronous and Multidimensional Synchronous Dataflow*, Ph.D. Thesis, Technical Report UCB/ERL M96/79, EECS Department, University of California, Berkeley, CA 94720, December 1996.
(http://ptolemy.eecs.berkeley.edu/papers/96/murthyThesis)

[77] T. M. Parks, *Bounded Scheduling of Process Networks*, Ph.D. Thesis, Technical Report UCB/ERL-95-105. EECS Department, University of California. Berkeley, CA 94720, December 1995.
(http://ptolemy.eecs.berkeley.edu/papers/parksThesis)

[78] S. Sriram, *Minimizing Communication and Synchronization Overhead in Multiprocessors for Digital Signal Processing,* Ph.D. Thesis, Tech. Report UCB/ERL 95/90, Dept. of EECS, University of California, Berkeley, CA 94720, November 7, 1995.
(http://ptolemy.eecs.berkeley.edu/papers/sriramThesis)

7.6 MASTERS REPORTS

[79] M. J. Chen, "Developing a Multidimensional Synchronous Dataflow Domain in Ptolemy," MS Report, Technical Report UCB/ERL No. 94/16, University of California, Berkeley, CA 94720, May 6, 1994.
(http://ptolemy.eecs.berkeley.edu/papers/mdsdfDomain.ps.Z)

[80] W. Chen, *Real-time Signal Processing on the Ultrasparc*, MS Report, Technical Report UCB/ERL No. 97/4, University of California, Berkeley, CA 94720, May 6, 1997.
(http://ptolemy.eecs.berkeley.edu/papers/97/ultrasparc)

[81] A. Peevers, *A Real-Time 3D Signal Analysis/Synthesis Tool Based on the Overlap-Add Short-Time Fourier Transform*, MS Report, Plan II, University of California, Berkeley, CA 94720, February 24, 1994.

[82] F. Sheikh, *Visualizing Architecture and Algorithm Interaction in Embedded Systems*, MS Report, EECS Department, University of California. Berkeley, CA 94720, September 17, 1996.
(http://ptolemy.eecs.berkeley.edu/papers/visualizing)

[83] S.-I. Shih, "Code Generation for VSP Software Tool in Ptolemy," MS Report, Plan II, Technical Report UCB/ERL M94/41, University of California, Berkeley, CA 94720, May 25, 1994.

[84] P. Warner, *Network of Workstations Active Messages Target for Ptolemy C Code Generation*, MS Report, Technical Report UCB/ERL No. 97/8, University of California, Berkeley, CA 94720, January 24, 1997.
(http://ptolemy.eecs.berkeley.edu/papers/97/now)

7.7 NEWSLETTER ARTICLES

[85] The Ptolemy Team, "System-Level Design Methodology for Embedded Signal Processors," *RASSP Digest Newsletter*, July, 1996. (http://ptolemy.eecs.berkeley.edu/papers/newsletters/)

[86] The Ptolemy Team, "The Ptolemy Kernel— Supporting Heterogeneous Design," *RASSP Digest Newsletter*, vol. 2, no. 1, pp. 14-17, 1st Quarter, April, 1995. (http://ptolemy.eecs.berkeley.edu/papers/newsletters/)

[87] The Ptolemy Team, "Ptolemy Seamlessly Supports Heterogeneous Design," *RASSP Enterprise Newsletter*, vol. 1, no. 3, August 1994.